# Tyabandha Journal of Arts and Science

**ISSN 1744–9669**

**Volume 4, Number 3**

**July 2007**

Edited by Vaen Sryayudhya

England
3rd February, 2008

Distribution
Internet-searchable world-wide

# Coin Grouping Problem

Kit Tyabandha, Ph.D.

## Abstract

We look at a coin grouping problem where there is a linear arrangement of coins in odd number and alternately head and tail.

## Introduction

'Coin Grouping' is a generic name. The problem we consider here is the case where there is an alternation between head and tail in an odd-numbered sequence of coins. In its simplest and non-trivial form, there are five coins. I first came across this problem of five-coin grouping around 1989. I soon found a solution which, interestingly, is algorithmic and generic for all odd numbers of coins. But it was not until recently that I tried to write the algorithm in a precisely fashion, the result of which is this present article.

## Statement of the Problem

**Problem 1.** There are an odd number of coins arranged alternately head and tail into a sequence. Our aim is with a sequence of moves to put all the heads together to one side, and all the tails to the other. Each of these moves must be a pair of exactly one head and one tail. A pair thus moved must touch some other coin.

§

From Problem 1 since the number of the coins is odd, either the heads or the tails must exceed in number the other by one. Without losing generality we choose to consider the case where there are more heads than tails. The case where there are five coins is represented as follows, H being a head and T a tail.

HTHTH

Assuming that all coins are of exactly the same size, the positions of them are discrete. We represent these by integers as shown in the following.

$$\begin{array}{ccccccccc} \cdots & 0 & 1 & 2 & 3 & 4 & 5 & 6 & \cdots \\ & & \mathrm{H} & \mathrm{T} & \mathrm{H} & \mathrm{T} & \mathrm{H} & & \end{array}$$

We write this as $H_1T_2H_3T_4H_5$.

## The Solution

**Definition 1.** A *cluster* is a chain of coins with no breaks therein.

In fact at each instance there can be at the most three clusters.

**Definition 2.** A *pivot* is the rightmost head in the coin sequence when all the coins form a single cluster.

The pivot divides the space, in this case the line, into two sides, left and right.

The following points summarise the solution here presented.

1. All moves toward the right are $HT$'s, those toward the left are $TH$'s.
2. A pivot is decided at each instance where there is one and only one cluster, and is used as *the* pivot until we should come across another such instance.
3. All pairs of coins are to be moved over from the left side to the right.

When there are more than one clusters, one needs to move pairs alternately between the left and the right sides until the pivot becomes the leftmost coin, at which point the then rightmost head becomes a new pivot.

## Example

**Example 1.** (Five Coins)

In the case of five coins, starting from $H_1T_2H_3T_4H_5$ our solution passes through the following steps in order, $H_3T_4H_5H_6T_7$, $H_5H_6T_7H_8T_9$, $H_5H_8T_9H_{10}T_{11}$, $H_5T_6H_7H_8T_{11}$, and $H_7H_8H_9T_{10}T_{11}$.

## An Algorithm

## Concluding Remarks

Counting the loops to determine the time-complexity one finds the number of steps taken to sort the coins as required to be 0, 1, 5, 14 and 30 steps for respectively 1, 3, 5, 7 and 9 coins. Tabulating this we find for five coins Table 1, for seven coins Table 2, and for nine coins Table 3.

| | |
|---|---|
| 2 | 1 |
| | 1 |
| | 1 |

**Table 1** *Counting the number of steps to sort five coins*

| | | |
|---|---|---|
| 3 | 2 | 1 |
| | 2 | 1 |
| | 2 | 1 |
| | 1 | |
| | 1 | |

**Table 2** *Counting the number of steps to sort seven coins*

| | | | |
|---|---|---|---|
| 4 | 3 | 2 | 1 |
| | 3 | 2 | 1 |
| | 3 | 2 | 1 |
| | | 2 | 1 |
| | | 2 | 1 |
| | | 1 | |
| | | 1 | |

**Table 3** *Counting the number of steps to sort seven coins*

# C Programming on Unix Part II

Kit Tyabandha, Ph.D.

Lecture

19th December 2006

Strịnondåḅurị

Library function usage

**Example 1.** (Read and write texts to files)

```
/* read, write and upper-case texts of file,
   Kit Tyabandha, 19 Dec 06 */
#include<ctype.h>
#include<stdio.h>
#include<stdlib.h>
void
filerror(char s[]){
  printf("\n Error : Can not open file ‘%s’\n\n", s);
}
void
puttext(char s[]){
  FILE *fpt;
  char c;
  fpt =fopen(s, "w");
  printf("\n Enter the contents for ‘%s’\
   (backslash to end)\n\n", s);
  do
    putc(c=getchar(), fpt);
  while (c!='\\');
  fclose(fpt);
}
```

```
void
showtext(char s[]){
  FILE *fpt;
  char c;
  if((fpt=fopen(s, "r"))==0)
    filerror(s);
  else{
    printf("\n\t Contents of file ‘%s’\n\n", s);
    do
      putchar(c=getc(fpt));
    while(c!='\\');
  }
  fclose(fpt);
}
void
uppercasetext(char s[], char S[]){
  FILE *fpt, *fpu;
  char c;
  short i =0;
  if((fpt=fopen(s, "r"))==0){
    filerror(s); i++;}
  if((fpu=fopen(S, "w"))==0){
    filerror(S); i++;}
  if(~i)
    do
      putc(toupper(c=getc(fpt)),fpu);
    while(c!='\\');
  fclose(fpt);
  fclose(fpu);
}
int
main(){
  puttext("data.txt");
  showtext("data.txt");
  uppercasetext("data.txt","DATA.txt");
  showtext("DATA.txt");
  exit(0);
}
```

Example 1 lists a programme that exchanges texts between standard input, standard output and a file, and transforms the same and puts in another file. The output of the programme is listed in Output 1.

**Output 1** *Compilation and run of Example 1*

```
kit@nebula:~/prog/c$ make tst
gcc -c -g -Wall tst.c
gcc -g tst.o -o tst -lcurses -ldl -lm
kit@nebula:~/prog/c$ tst
 Enter the contents for ‘data.txt’ (backslash to end)
The road goes ever on and on
Down from the door where it began.
                J.R.R. Tolkien
                (‘Lord of the Rings’, 1954) \
         Contents of file ‘data.txt’
The road goes ever on and on
Down from the door where it began.
                J.R.R. Tolkien
                (‘Lord of the Rings’, 1954) \
         Contents of file ‘DATA.txt’
THE ROAD GOES EVER ON AND ON
DOWN FROM THE DOOR WHERE IT BEGAN.
                J.R.R. TOLKIEN
                (‘LORD OF THE RINGS’, 1954)
```

□

Example 2 is a programme which shows how to manipulate integral arithmetic in various bases, namely decimal, hexadecimal and octal. Output 2 is its output.

**Example 2.** (Arithmetic of decimal, octal and hexadecimal integers)

```
#include<stdio.h>
int main(){
  short a, b, c;
  a =12;
  b =47;
  c =a+b;
  printf("\n\t Decimal\n\
    a = %d, b = %d, c = a+b = %d\n\n", a, b, c);
  a =012;
  b =047;
  c =a+b;
  printf("\n\t Decimal equivalent of octal\n\
    a = %d, b = %d, c = a+b = %d\n\
   (a = %o, b = %o, c = a+b = %o in octal)\n\n",\
      a, b, c, a, b, c);
  a =0x12;
  b =0x47;
  c =a+b;
  printf("\n\t Decimal equivalent of hexadecimal\n\
    a = %d, b = %d, c = a+b = %d\n\
   (a = %x, b = %x, c = a+b = %x in hexadecimal)\n\n",\
      a, b, c, a, b, c);
  return 0;
}
```

¶

**Output 2** *Output of Example 2*

```
         Decimal
    a = 12, b = 47, c = a+b = 59
         Decimal equivalent of octal
    a = 10, b = 39, c = a+b = 49
   (a = 12, b = 47, c = a+b = 61 in octal)
         Decimal equivalent of hexadecimal
    a = 18, b = 71, c = a+b = 89
   (a = 12, b = 47, c = a+b = 59 in hexadecimal)
```

□

Example 3 shows a programme demonstrating the use of the power function `pow()`, which takes two inputs. The output is given in Output 3.

**Example 3.** (Usage of the power function)

```
/* power function, a library function, Kit Tyabandha,
   20 Nov 06*/
#include<math.h>
#include<stdio.h>
int main(){
  int d1, d2, p;
  d1 =4;
  d2 =3;
  p =pow(d1,d2);
  printf("\n %d to the power of %d is %d\n\n", d1, d2, p);
  return(0);
}
```

¶

**Output 3** *Output to Example 3*

```
 4 to the power of 3 is 64
```

□

Example 4 is a programme to study some of the mathematical functions. The output is shown as Output 4. Notice how mathematical constants like Pi ($\pi$) which is infinite in size are kept on the computer as floating point constants. Also $\sin(60°) = \cos(30°) = \sqrt(3)/2$ and some hyperbolic functions, namely sinh and cosh.

**Example 4.** (Study of some mathematical functions)

```
/* mathematical functions, Kit Tyabandha, 19 Dec 06 */
#include<math.h>
#include<stdio.h>
int main(){
  printf("\n Pi is %f\n", M_PI);
  printf("\n Pi is %.9f\n", M_PI);
  printf("\n Pi is %.60f\n", M_PI);
  printf("\n sine(pi) = %f\n", sin(M_PI));
  printf("\n sine(30 degree) = %f\n", sin(30*M_PI/180));
  printf("\n cosine(30 degree) = %f\n", cos(30*M_PI/180));
  printf("\n sine(60 degree) = %f\n", sin(60*M_PI/180));
  printf("\n sqrt(3)/2 = %f\n", sqrt(3)/2);
  printf("\n sinh(pi/3) = %f\n", sinh(M_PI/3));
  printf("\n cosh(pi/3) = %f\n", cosh(M_PI/3));
  return(0);
}
```

¶

**Output 4** *Output of Example 4*

```
Pi is 3.141593
Pi is 3.141592654
Pi is 3.141592653589793115997963468544185161590576171875000000000000
sine(pi) = 0.000000
sine(30 degree) = 0.500000
cosine(30 degree) = 0.866025
sine(60 degree) = 0.866025
sqrt(3)/2 = 0.866025
sinh(pi/3) = 1.249367
cosh(pi/3) = 1.600287
```

□

Arrays can be multi-dimensional. An example of a three-dimensional array is demonstrated with the use of a programme in Example 5, which gives an output shown in Output 5.

**Example 5.** (Three-dimensional array)

```
/* 3-d array's addresses, Kit Tyabandha, 28 Nov 2006 */
#include<stdio.h>
int main(){
  int i, j, k, a[2][3][4];
  for(i=0; i<2; i++){
    for(j=0; j<3; j++){
      for(k=0; k<4; k++){
        printf("&a[%d][%d][%d] = %x\n", i, j, k, &a[i][j][k]);
      }
    }
  }
  return 0;
}
```

¶

**Output 5** *Output to Example 5*

```
&a[0][0][0] = bffffa80
&a[0][0][1] = bffffa84
&a[0][0][2] = bffffa88
&a[0][0][3] = bffffa8c
&a[0][1][0] = bffffa90
&a[0][1][1] = bffffa94
&a[0][1][2] = bffffa98
&a[0][1][3] = bffffa9c
&a[0][2][0] = bffffaa0
&a[0][2][1] = bffffaa4
&a[0][2][2] = bffffaa8
&a[0][2][3] = bffffaac
&a[1][0][0] = bffffab0
&a[1][0][1] = bffffab4
&a[1][0][2] = bffffab8
&a[1][0][3] = bffffabc
&a[1][1][0] = bffffac0
&a[1][1][1] = bffffac4
&a[1][1][2] = bffffac8
&a[1][1][3] = bffffacc
&a[1][2][0] = bffffad0
&a[1][2][1] = bffffad4
&a[1][2][2] = bffffad8
&a[1][2][3] = bffffadc
```

□

## Lecture

26th December 2006

Strịnondảḅurị

# Unix and Knoppix

After its birth and up until AT& T's Version 6 Unix had been very useful in Computer Science courses on Operating Systems since the source code could be freely distributed, discussed and taught. With the arrival of Version 7, arguably the best one ever, the availability of the code was discontinued. This led to Andrew Tanenbaum who was a professor teaching Computer Science in the Netherlands, writing from scratch in 1987 his own code in C for Minix (Mini Unix). Having written a book in which he gave all the code for the new operating system he used it for his own teaching and research and, together with his research students produced Minix 2 in 1997 and Minix 3 in 2004. (Tanenbaum and Woodhull, 2006)

The purpose of Minix was to demonstrate how an operating system work. A computer science student whose name was Linus Torvalds began first by using Minix and writing his own drivers and so on, and then decided to write his own code for an operating system to do what Unix does and to be more efficient than Minix. His aim was not for educational purpose but for real use. Thus came Linux 0.01 in August 1991 and Linux 1.0 on 13 March 1994.

Linux work force has formed into various groups, to mention but two Debian and Red Hat. All in Unix family, including Minix and Linux, follow the standard set up by IEEE called POSIX (Portable Operating System Interface).

Knoppix is a live-CD system based on Debian. Here we chose Knoppix 3.9 because it is efficient and has both `emacs` and `gcc` needed for the course. The Knoppix CD can also be installed on to a computer. You would only do this if you want to replace all the existing systems on your computer with Knoppix's Debian.

## Emacs

The programme Editor Macros (Emacs) was written by Richard Stallman. It was him who founded both the Free Software Foundation and the GNU project. Emacs commands in normal use are in the form of key sequences. All of these commands are actually short cuts of some full-text commands, which are seldom used except in cases where no equivalent key sequence exists. One also finds combinations of key sequences and full-text commands, for example `M-x replace-string`.

Tabble 1 gives some of the key sequences in common use. Here the prefixes `C` and `M` means respectively a control key and a meta (escape) key. With the key sequences like `C-x C-f` for opening a file the control key can be pressed and kept down while the other two keys are pressed one after the other. While the control key is kept down, however, the meta key is released before its partner key is pressed, for example `M-v` for scrolling up is `M` first and then `v`.

| *Function* | *Key sequence* | *Description* |
|---|---|---|
| Cursor | `C-a` | cursor to beginning of the line |
| | `C-b` | move cursor left |
| | `C-e` | cursor to end of the line |
| | `C-f` | move cursor right |
| | `C-n` | move cursor down |
| | `C-p` | move cursor up |
| | `M-b` | move backwards one word |
| | `M-f` | move forwards one word |

**Table 1** *Emacs key sequences*

| *Function* | *Key sequence* | *Description* |
|---|---|---|
| Edit | `C-d` | delete a character |
| | `C-k` | erase to end of the line |
| | `C-t` | swap two adjacent letters |
| | `C-x C-i` | read in the contents of a file |
| | `C-x C-r` | reread a file |
| | `C-x C-t` | swap two lines |
| | `C-o` | insert a line at cursor position |
| | `C-w` | delete to mark |
| | `C-y` | yank |
| | `C-_` | undo the previous change |
| | `M-c` | capitalise initial letter of the word |
| | `M-d` | delete a word |
| | `M-l` | lowercase to end of word |
| | `M-t` | swap two words |
| | `M-u` | uppercase to end of word |
| | `M-%` | query replace |
| Marking | `C-Spc` | set mark |
| Programme | `C-g` | quitting the current action in preparation |
| | `C-x C-c` | exit from the programme |
| | `C-x C-f` | open a file |
| | `C-u M-!` | enter shell commands, output into this buffer |
| | `M-!` | enter shell commands, output to another buffer |
| Saving | `C-x C-s` | save the current buffer |
| | `C-x C-w` | save current buffer to a specified file |
| Scrolling | `C-l` | scroll to put current cursor position in middle |
| | `C-x [` | scroll backwards one page |
| | `C-x ]` | scroll forwards one page |
| | `C-v` | scroll down one screen |
| | `M-v` | scroll up one screen |
| | `M-<` | goto beginning of the file |
| | `M->` | goto end of the file |
| Search | `C-r` | search backwards for a specified string |
| | `C-s` | search forwards for a specified string |

**Table 1**(continued) *Emacs key sequences*

The sequence `M-\%` queries and replaces, while `M-x replace-string` replaces all instances met without asking for confirmation. While replacing thus `y` does the replacing and continues to the next occurrence while `n` does no replacing and continue. Also, `!` can replace the rest without asking, `?` gives a list of options, `.` replaces the current instance and quits, `,` replaces the current instance but does not move on to the next one.

## Knoppix

The philosophy of Knoppix is to allow as little write access as possible. But one can issue the command `su` and then mount a floppy drive using the following command (assuming the drive is logically on /dev/fd0).

```
mount -t msdos -w /dev/fd0 /floppy
```

Similarly one may also mount a hard disk or a partition using, for example,

```
mount -w /mnt/hda5
```

Here we have prior to this checked (by examining the device icon on desktop) that the partition we want to work on is `/mnt/hda5`.

## Unix

Table 2 lists some of the Unix commands that are frequently used together with their description. To read the manual page for a command, type `man` followed by the command's name. Typine a command's name followed by `--help` gives a brief description of the options available. To see what `man` does, say, type `man man`. Most relevant to this course, study `man emacs` and `man gcc` to learn more about both `emacs` and `gcc` we use.

With Unix everything is represented as a file. This includes things like directories and devices.

| *Command* | *Description* |
|---|---|
| `awk` | pattern scanning and processing language |
| `bc` | a calculator language |
| `cal` | a calendar |
| `calendar` | reminder service |
| `cat` | concatenate files and print on standard output |
| `cp` | copy files |
| `cut` | removes sections from each line of a file |
| `dc` | a calculator in Reverse Polish Notation |
| `df` | reports file system disk space usage |
| `dmesg` | print or control the kernel ring buffer |
| `du` | gives memory usage of each file |
| `emacs` | an extensible editor |
| `find` | search for files |
| `grep` | prints lines matching a pattern |
| `less` | file perusal filter for viewing on CRT |
| `ls` | list directory contents |
| `man` | on-line reference manual |
| `mkdir` | make a directories |
| `more` | file perusal filter for viewing on CRT |
| `mount` | mount a file system |
| `mv` | move files |
| `nl` | number lines in files |
| `rmdir` | remove empty directories |
| `sed` | a stream editor |
| `sort` | sort lines of text files |
| `su` | change UID to that of the superuser (root) or another user |
| `tar` | archives files |
| `tee` | reads from standard input, write to standard output and files |
| `touch` | create a blank file |
| `umount` | unmount a file system |
| `uniq` | removes duplicate lines from a sorted file |
| `vi` | basic text editor |
| `wc` | gives the number of newlines, words and bytes in files |

**Table 2** *Unix commands*

A *regular expression* is an expression that uses special characters as masks for file names. Thus the result is a set of a number of files which meet the masking. For example,

```
ls lond06a0{0[1-9],[1-6][0-9]}.jpg
```

will list all the files from `lond06a001.jpg` to `lond06a069.jpg` whereas,

```
ls lond*.jpg
```

will list all files beginning with `lond` and ending with `.jpg`.

The text editor `vi` is basic, which means that it is very important to learn how to use. Likelier than not, in dire situations when everything else fails it would be the only thing to work. So it is essential for a programmer to know.

As an example of `awk` try the following.

```
awk -F: '{print $5 $3}' /etc/passwd
```

`dmesg` is used to list the boot messages. We can redirect the output of this command into a file by `dmesg >` *file*.

Piping allows one process to transfer its output to another. The symbol for a pipe is simply a vertical bar. The following finds all `.txt` files whose name contains the string “cpg”.

```
find ./ *.txt | grep cpg
```

Some command line tips, `C-p` brings up the previous command entered, `C-n` brings up the following command, and `C-u` clear to beginning of the line.

## Bibliography

Jack Tackett, Jr and Steven Burnett. *Special Edition Using Linux.* 4th ed., Que, 1999

Andrew S Tanenbaum and Albert S Woodhull. *Operating Systems Design and Implementation.* 3rd ed., Prentice-Hall, 2006

## Lecture

28th December 2006

Strịnondȧḅurị

## Example programme

The following is a programme to help with the test.

```
/* programme to help with the test, Kit Tyabandha, 28 Dec 06 */
#include<stdio.h>
#include<stdlib.h>
int num=23531, sn=27;
void
questions(){
  FILE *fpt;
  int i, j, k, qe=5, qn=65;
  int chk[qn], qst[sn][qe];
  for(i=0; i<sn; i++){
    for(j=0; j<qn; j++) chk[j]=0;
    for(j=0; j<qe; j++){
      while(chk[k=rand()%qn]);
      chk[k]++;
      qst[i][j] =++k;
    }
  }
  fpt =fopen("qsts.txt", "w");
  fprintf(fpt, "ID\tQ1\tQ2\tQ3\tQ4\tQ5\n\n");
  for(i=0; i<sn; i++){
    fprintf(fpt, "%d", i+1);
    for(j=0; j<qe; j++){
      fprintf(fpt, "\t%d", qst[i][j]);
    }
    fprintf(fpt, "\n");
  }
  fclose(fpt);
}
void
reorderids(){
  FILE *fpt;
  int i;
  int id[sn], rnd[sn];
  for(i=0; i<sn; i++){
    id[i] =i+1;
    rnd[i] =rand();
  }
  fpt =fopen("id.txt", "w");
  for(i=0; i<sn; i++){
    fprintf(fpt, "%d %d\n", id[i], rnd[i]);
  }
  fclose(fpt);
  system("sort -k2 id.txt | cut -d' ' -f1 | nl > idd.txt");
}
int main(){
  srand(num);
  reorderids();
  questions();
  return 0;
}
```

¶

## Test 1 (20%)

28th December 2006

**A.** Questions related to Unix. Explain and do the following at the shell prompt.

**1.** list contents of the directory
**2.** list files in long format
**3.** find out the present working directory
**4.** make a new directory
**5.** remove an existing directory
**6.** change UID to that of the superuser
**7.** peruse the manual page of a command
**8.** print lines in a file that match a pattern
**9.** mount a file system
**10.** mount a file system for read and write
**11.** unmount a mounted file system
**12.** change the working directory
**13.** move files
**14.** copy files
**15.** use the command `awk`
**16.** search for files
**17.** report file-system disk-space usage
**18.** reminder service for memorable events
**19.** create a blank file
**20.** concatenate files
**21.** count the number of words in a file
**22.** look at the boot message
**23.** print the kernel ring buffer
**24.** sort lines of a text file
**25.** redirect a command's output into a file
**26.** put a regular expression to use
**27.** use a calculator
**28.** list all `.c`'s files in the directory
**29.** prepare for a shutdown
**30.** bring down the system

**B.** Questions related to Emacs. Explain and do the following.

**31.** move cursor to the beginning of the line
**32.** move cursor right
**33.** move cursor left
**34.** move cursor to the end of the line
**35.** move cursor up
**36.** move cursor down
**37.** move cursor forward one word
**38.** delete a character
**39.** erase to end of line
**40.** read in (insert) the contents of a file
**41.** insert a line at cursor position
**42.** set a mark
**43.** delete to mark
**44.** yank
**45.** undo the previous change
**46.** query and replace
**47.** discontinue the current action
**48.** open a file
**49.** save the current buffer
**50.** do shell commands from within Emacs
**51.** put cursor position in the vertical middle
**52.** scroll forward one page
**53.** scroll backward one page
**54.** scroll down one screen
**55.** scroll up one screen
**56.** goto the beginning of the buffer (file)
**57.** goto the end of the buffer (file)
**58.** search forward for a specified string
**59.** search backward for a specified string
**60.** exit from the programme

**C.** Questions related to GCC. Explain and do the following.

**61.** Write a make file.

Consider as an example the following make file where `tst.c` is the name of our source file.

```
cc=gcc
cflags=-c -g -Wall
lflags=-g
libs=-lcurses -ldl -lm
tst : tst.o
	${cc} ${lflags} $< -o tst ${libs}
%.o : %.c
	${cc} ${cflags} $<
```

Which says that the C compiler used is GCC. The flag `-c` says that the source codes are to be compiled and assembled, but not linked. The flag `-g` gives information that could be used for debugging. You can use `gdb` to debug using this information. The flag `-Wall` turns on all warnings. Line indentations must be produced by tabs only.

For the following, compile with `gcc -o file file.c`, where `file` is the file's name, then run the executable file.

**62.** Write a programme to print out some messages on the standard output.

Here we use windows I/O instead of regular input and output.

```
/* Extract from Chaucer's Canterbury Tales, Kit Tyabandha,
   30 Jan 07 */
#include<curses.h>
#include<stdio.h>
int main(){
  initscr();
  move(7,20);
  printw("But ther been folk of swich condicion\n\
    \t\t That whan they have a certein purpose take,\n\
    \t\t They kan nat stynte of hire entencion,\n\
    \t\t But, right as they were bounden to a stake,\n\
    \t\t They wol nat of that firste purpos slake.");
  move(15,35);
  addstr("Canterbury Tales (c. 1387)");
  move(17,35);
  addstr("Geoffrey Chaucer (c. 1340-1400)");
  box(stdscr, '; '-');
  getch();
  endwin();
  return 0;
}
```

**63.** Write a programme to calculate $c = a + b$, where $a$ and $b$ are two variables with a specified numerical value.

An answer for a minimalist could be the following.

```
#include <stdio.h>
int
main(){
  int a, b=2, c=3;
  a =b+c;
  printf("\n b = %d, c = %d, a = b+c = %d\n\n", b, c, a);
  return 0;
```

**64.** Write a programme to calculate $c = a\%b$, where $a$ and $b$ are two specified integral variables.

```
#include <stdio.h>
int
main(){
  int a, b=11, c=3;
  a =b%c;
  printf("\n b = %d, c = %d, a = b %% c = %d\n\n", b, c, a);
  return 0;
}
```

**65.** Write a programme to create and write into a file "Hello World!"

```
#include <stdio.h>
int
main(){
  FILE *fpt;
  fpt =fopen("tmp.txt", "w");
  fprintf(fpt, "\n Hello world!\n\n");
  fclose(fpt);
  return 0;
}
```

## Lecture

9th January 2006

Strịnondȧḅurị

## Problem-oriented programming

All programmes exist to serve some purposes and to solve some problems. Inherently thus they are problem-oriented. At the same time, no solutions to any problem are perfect. This means that all programmes can be improved. It is therefore important for us to keep our mind on the problems and not on any particular solution.

Problem 1 and what follow is a case study of such problem as we may write our programmes to help answer. Here the subject for our test is this present course we are learning, namely C and Unix.

**Problem 1.** How to have a fair test?

Problem 1 may be said to be *the* problem in this case. From it spring loads of other problems to consider, both detailed and practical ones. One such problem is how to make a test that is fair to *this* subject.

Since both C and Unix can be considered a skill or a language, we conclude that our attention should be on hands-on experience. But it is one thing to write an answer to a question correctly, and quite another to carry it out in actual practice. Also there is a hazard of answers obtained by asking or telling friends, which could render a test unfair and useless. Keeping all this in mind we choose for our test an oral exam of the 'show-me how-to' type. Then it is important that each person being tested has a whole attention of the one carrying out the test. Therefore we choose for us a sequential test where questions are asked one person after another.

Ideally in such a test one needs to make sure that each person get a different set of questions from others. Also there is an issue of the fairness of the choice of questions in each set. This is impossible since any two questions are different. For now we find our way out of this dilemma by having the questions in the sets randomly chosen.

But randomness as implemented by the library function `rand` in C is in fact merely a very long sequence of integers. Though as a whole the sequence is practically without any patterns, yet it is a known sequence. Which means that if we know where in the sequence we begin we will be able to predict our sequence of numbers by simply reading off tables. Then our sequence of numbers would not be very random after all, in the sense that we could find out where it is going next.

But all is not yet hopeless. We still could have our chosen sequence random by choosing our starting point on the `rand` sequence and make sure that *that* is randomly done.

In our case we ensure this randomness in our choice of where in that sequence to begin our sequence by having each student choose an integer which are then added and used as the so-called *seed*, that is our starting point.

All this represents our solution and the way we carry out our test. Algorithm 1 puts together what we have said up to now.

```
$seed \leftarrow 0$
```

```
for each student do
  choose a number, num
  seed ← seed + num
endfor
initialise rand() with seed
choose randomly the order by which students are to be tested
for each student do
  choose randomly five questions from the set of questions
endfor
```

Numbers chosen by students are in their usual order 3, 5, 9, 7, 4, 4, 4, 7, 11, 3, 93, 33, 7, 7, 5, 9, 9, 15, 7, 9, 2, 5, 7, 8, 9, 2 and 4. They add up to 288, and this number was used as a seed for our random number generation. Next, the random numbers generated from the seed are put into *id.txt*, which is given in Listing 6.

**Listing 6** *Random numbers for students*

```
1 1492921876
2 2031279135
3 1374965837
4 363701165
5 1023699735
6 1225876319
7 961810921
8 2016580502
9 968848298
10 1835762276
11 1490431874
12 197035131
13 89380432
14 1876165494
15 1164775919
16 717716400
17 1636411253
18 135000917
19 239069903
20 1216396086
21 1688011284
22 1256455512
23 1854796271
24 1758355444
25 192848381
26 1049089232
27 722243584
```

The ordering of students extracted from *idd.txt* is 5, 26, 15, 20, 6, 22, 18, 3, 11, 1, 17, 21, 24, 10, 23, 14, 25, 12, 8, 2, 19, 4, 16, 27, 13, 7 and then 9.

Finally, the five questions assigned to each student are shown for all students in Listing 7, which is the contents of *qsts.txt*.

**Listing 7** *Five questions to each student*

```
ID Q1 Q2 Q3 Q4 Q5
1 1 20 7 26 13
2 24 65 48 7 44
3 55 36 10 28 45
4 56 1 16 25 53
5 1 37 21 64 63
6 8 16 14 19 40
7 63 21 59 7 46
8 9 32 46 56 38
9 26 47 8 35 12
10 54 26 14 6 50
```

**Listing 7** *(continued)*

```
11 1 7 24 21 5
12 23 31 22 37 51
13 64 36 7 57 42
14 54 2 9 34 58
15 48 60 41 56 29
16 54 46 56 2 52
17 43 2 60 1 25
18 64 23 55 61 40
19 21 32 48 14 10
20 37 16 20 5 10
21 3 1 52 60 32
22 41 40 22 42 26
23 64 44 20 1 3
24 21 26 59 43 35
25 65 54 18 13 64
26 54 28 18 60 39
27 22 63 26 16 29
```

Table 3 gives a list of students together with the marks and score from their test.

| *Number* | *Student* | | | | *Marks* | | *Total* |
|---|---|---|---|---|---|---|---|
| 1 | Fon | 2 | 1 | 1 | 1 | 1 | 6 |
| 2 | Mook | 2 | 1 | 1 | 4 | 4 | 12 |
| 3 | Eiw | 4 | 4 | 1 | 1 | 1 | 11 |
| 4 | Re | 2 | 4 | 3 | 4 | 1 | 14 |
| 5 | Meam | 2 | 2 | 2 | 1 | 2 | 9 |
| 6 | Oum | 1 | 2 | 1 | 2 | 3 | 9 |
| 7 | Jan | 4 | 2 | 4 | 4 | 2 | 16 |
| 8 | Film | 4 | 4 | 2 | 4 | 4 | 18 |
| 9 | Pear | 2 | 4 | 2 | 4 | 2 | 14 |
| 10 | Yo | 4 | 1 | 2 | 4 | 1 | 12 |
| 11 | Na | 4 | 2 | 2 | 2 | 4 | 14 |
| 12 | Nest | 4 | 4 | 2 | 2 | 2 | 14 |
| 13 | Tang | 2 | 4 | 4 | 4 | 4 | 18 |
| 14 | Seven | 4 | 1 | 2 | 4 | 4 | 15 |
| 15 | Aim | 1 | 2 | 4 | 2 | 3 | 12 |
| 16 | AApi | 4 | 1 | 4 | 2 | 1 | 12 |
| 17 | AChu | 2 | 1 | 2 | 4 | 2 | 11 |
| 18 | Eua | 1 | 4 | 1 | 1 | 1 | 8 |
| 19 | Ja | 1 | 2 | 2 | 4 | 4 | 13 |
| 20 | Bo | 1 | 1 | 2 | 1 | 1 | 6 |
| 21 | Ta | 2 | 4 | 1 | 2 | 1 | 10 |
| 22 | Dawn | 4 | 1 | 1 | 4 | 1 | 11 |
| 23 | Nulek | 1 | 4 | 4 | 4 | 2 | 15 |
| 24 | Bee | 2 | 2 | 2 | 4 | 4 | 14 |
| 25 | Namtan | 1 | 4 | 2 | 2 | 1 | 10 |
| 26 | Ying | 4 | 1 | 1 | 1 | 1 | 8 |
| 27 | Gift | 1 | 1 | 2 | 2 | 1 | 7 |

**Table 3** `tsmsn`

Note that to compile Table 3 we use `awk` in Unix as follows.

```
awk '{printf "%d %s %d %d %d %d %d %d\n", $1, $3, $4, $5,\
    $6, $7, $8, $4+$5+$6+$7+$8}' ./test1.dat > tmp
```

Here the fields in `test1.dat` are *number, name, other name, score for question 1, 2, 3, 4* and *5*.

The frequency of each question being chosen are in the format *question*(*frequency*) in their order, 1(0), 2(7), 3(3), 4(2), 5(0), 6(2), 7(1), 8(5), 9(2), 10(2), 11(3), 12(0), 13(1), 14(2), 15(3), 16(0), 17(4), 18(0), 19(2), 20(1), 21(3), 22(5), 23(3), 24(2), 25(2), 26(2), 27(6), 28(0), 29(2), 30(2), 31(0), 32(1), 33(3), 34(0), 35(1), 36(2), 37(2), 38(3), 39(1), 40(1), 41(3), 42(2), 43(2), 44(2), 45(2), 46(1), 47(3), 48(1), 49(3), 50(0), 51(1), 52(1), 53(2), 54(1), 55(5), 56(2), 57(4), 58(1), 59(1), 60(2), 61(4), 62(1), 63(0), 64(3), and 65(5).

Programme 8 is written to mark Test 1.

**Programme 8** *Programme to mark the examination*

```
/* Programme to mark Test 1, Kit Tyabandha, 30 Dec 2006 */
#include<stdio.h>
struct student{
  int id;
  char nm[12];
  char nknm[6];
  int mrk[10];
};
void
score(){
  FILE *fpt;
  int i, j, mrkn=5, studn=27;
  int scr[studn];
  struct student stud[studn];
  for(i=0; i<studn; i++){
    scr[i] =0;
  }
  fpt =fopen("test1.dat", "r");
  for(i=0; i<studn; i++){
    fscanf(fpt, "%d %s %s", &stud[i].id, stud[i].nm,\
     stud[i].nknm);
    for(j=0; j<mrkn; j++){
      fscanf(fpt, " %d", &stud[i].mrk[j]); fscanf(fpt, "\n");
    }
  }
  fclose(fpt);
  for(i=0; i<studn; i++){
    for(j=0; j<mrkn; j++){
      scr[i] +=stud[i].mrk[j];
    }
  }
  fpt =fopen("score1.opt", "w");
  fprintf(fpt, "ID\tName\tScore\n\n");
  for(i=0; i<studn; i++){
    fprintf(fpt, "%d\t%s\t%d\n", stud[i].id,\
     stud[i].nm, scr[i]);
  }
  fclose(fpt);
}
int
main(){
  score();
  return 0;
}
```

Programme 9 finds frequency of occurrence of questions. The results of this are given above.

**Programme 9** *Programme to find frequency of questions*

```
/* find frequency of questions, Kit Tyabandha, 9 Jan 2007*/
#include<stdio.h>
#include<stdlib.h>
int
main(){
  char c[2];
  FILE *fpt;
  int i, j, q[5], qn=65, id, sn=27;
  int f[qn];
  for(i=0; i<qn; i++){
    f[i]=0;
  }
  fpt =fopen("qsts.txt", "r");
  fscanf(fpt, "%s\t%s\t%s\t%s\t%s\t%s\n", c, c, c, c, c, c);
  for(i=0; i<sn; i++){
    fscanf(fpt, "%d\t%d\t%d\t%d\t%d\t%d", \
      &id, &q[0], &q[1], &q[2], &q[3], &q[4]);
    for(j=0; j<5; j++){
      f[q[j]]++;
    }
  }
  fclose(fpt);
  fpt =fopen("foq.txt", "w");
  fprintf(fpt, "Question\tFrequency\n\n");
  for(i=0; i<qn; i++){
    fprintf(fpt, "%d\t%d\n", i+1, f[i]);
  }
  fclose(fpt);
  exit(0);
}
```

## Problem and possible solution

As the test was held in a large computer room where students are free to wander around, there is a question of whether those being tested later could learn or gain information from their less unfortunate friends. One possible solution to this is to carry out the test in a separate room and to have students take turns entering it one after another.

This would have an additional advantage that all the necessary materials could then be prepared in advance, for example a header file and a text file for students to demonstrate their solution on.

# An Ḋiaö, Un Dieu, One God Part II

Kit Tyabandha, Ph.D.

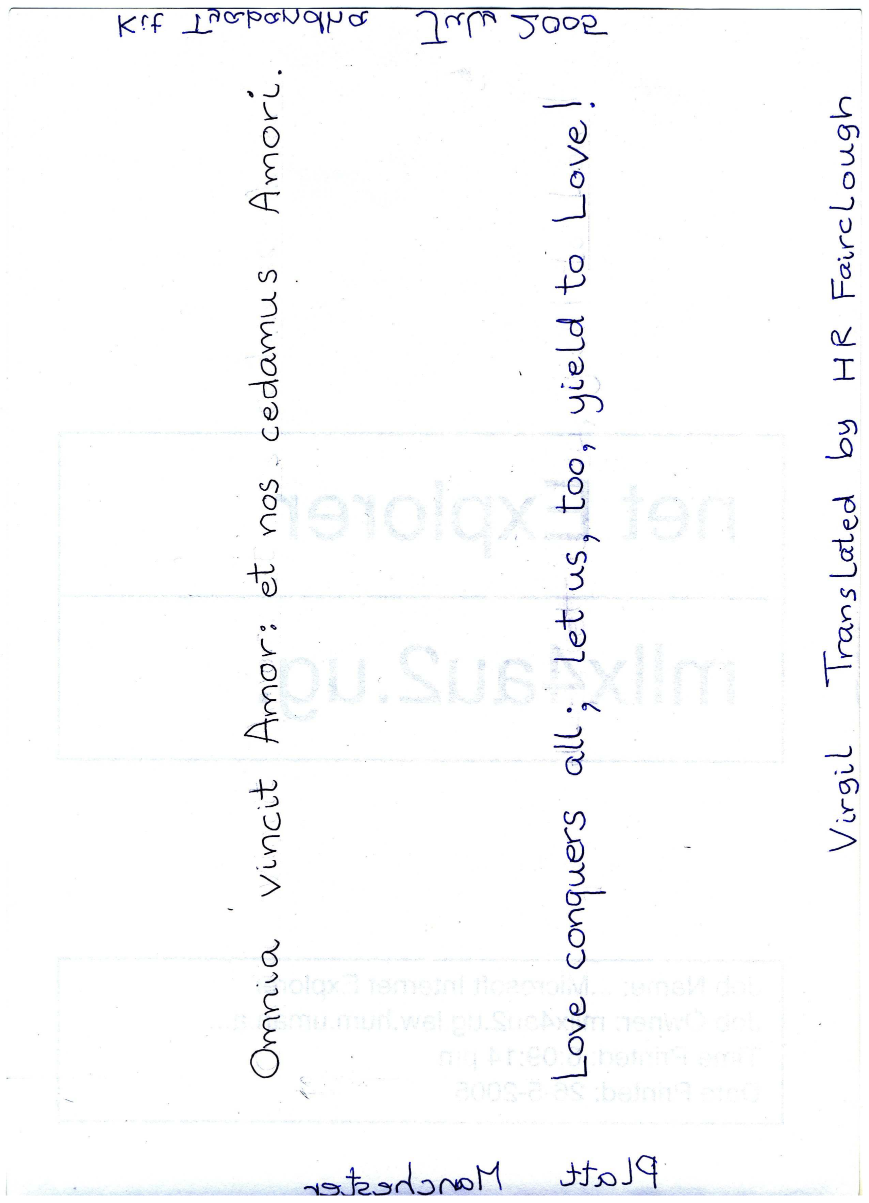
This art exhibition article talks about God, the one Father of Abraham, Jesus and Muhammad, and the Truth as meant by Buddha. It is a continuation of Part I with the same title, that appeared on Page 3 in the same volume of this journal.

The artist Kit Tyabandha has spent his life in various corners of the world. He has lived in England, Hungary, Japan and New Zealand. All his life he has been searching for God. These works represent a collection of his spiritual experience. Some of the works in this collection have been on display at art exhibitions in Tokyo. Others have been used in talks Tyabandha delivered in England. There are also those which have been made and used to describe some ideas to friends.

All materials used to produced these works are recycled. Header papers from public printers at University of Manchester get thrown away. I like to reuse them on their remaining blank side to reflect on questions in philosophy, and to describe God.

*An Ḋiaö* is a Thai phrase which means ‘the only one’. Similarly *Un Dieu* is French for ‘One God’. The meaning of the title of this proposed exhibition is the play on the similarity among these three which bridges cultural differences, as only God could.

Kit Tyabandha, Ph.D.
Bangkok, January 2007



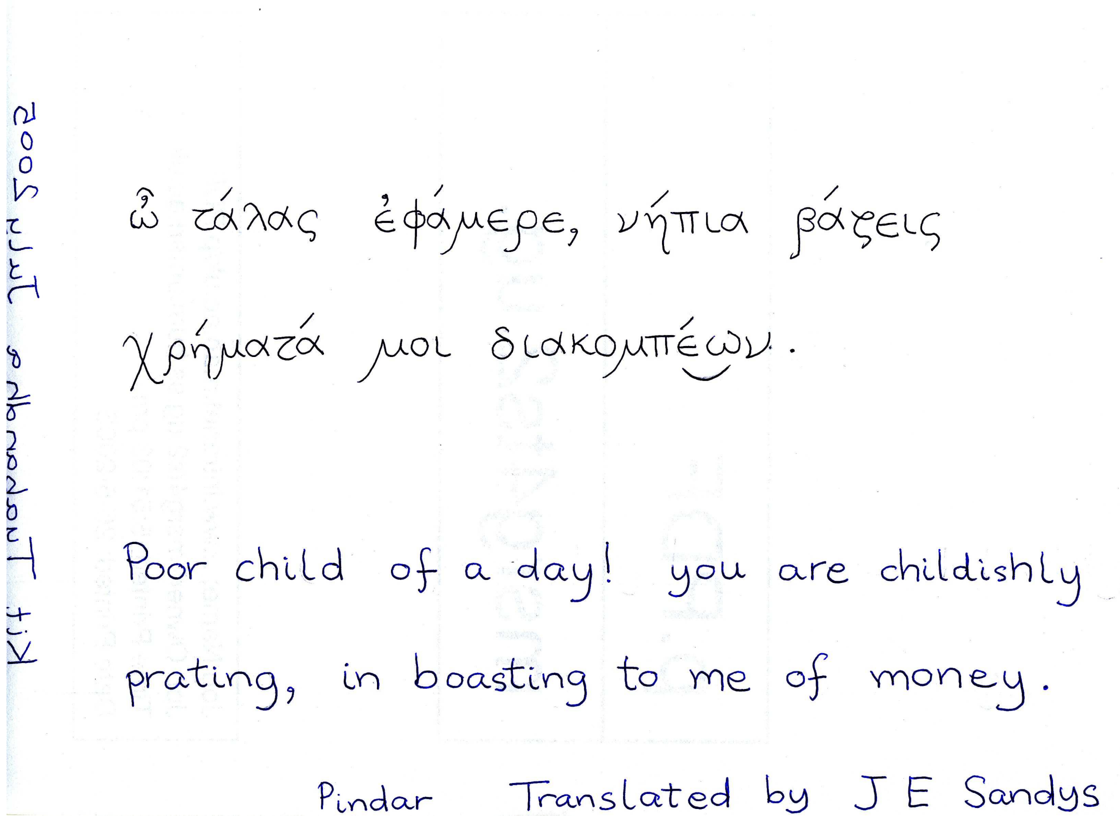
Kit Tyabandha Jul 2005

Omnia vincit Amor: et nos cedamus Amori.

Love conquers all; let us, too, yield to Love!

Virgil Translated by HR Fairclough

Platt Manchester

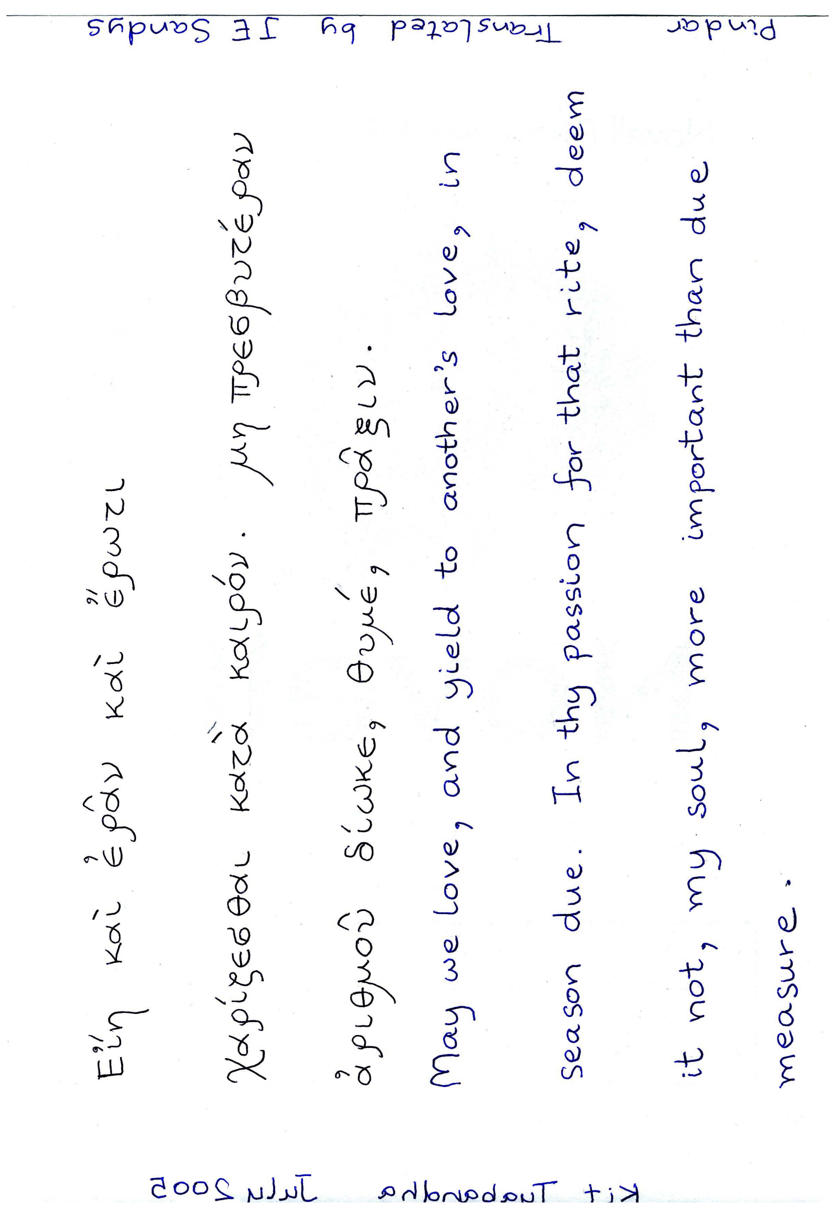


ὦ τάλας ἐφάμερε, νήπια βάζεις
χρήματά μοι διακομπέων.

Poor child of a day! you are childishly
prating, in boasting to me of money.

Pindar Translated by J E Sandys

Kit Tyabandha July 2005

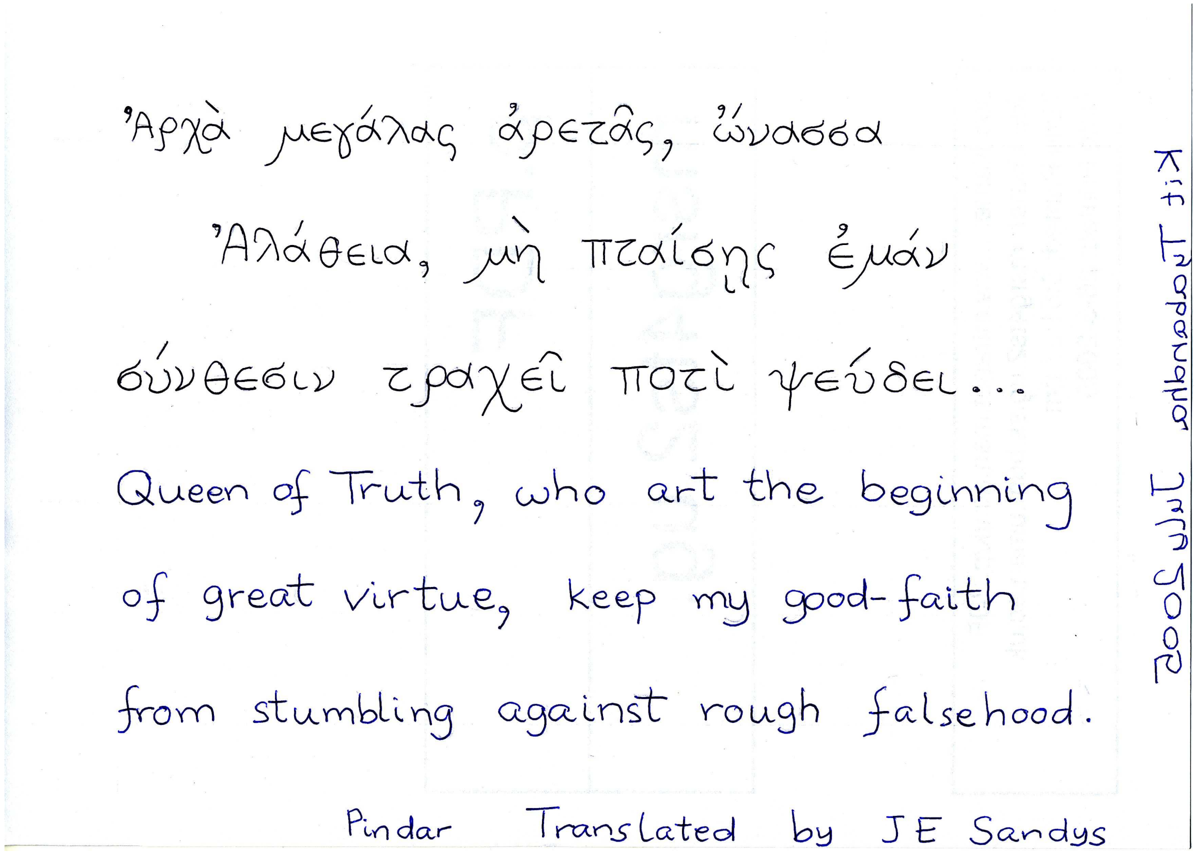


Pindar Translated by JE Sandys

Εἴη καὶ ἐρᾶν καὶ ἔρωτι
χαρίζεσθαι κατὰ καιρόν. μη πρεσβυτέραν
ἀριθμοῦ δίωκε, θυμέ, πρᾶξιν.

May we love, and yield to another's love, in
season due. In thy passion for that rite, deem
it not, my soul, more important than due
measure.

Kit Tyabandha July 2005

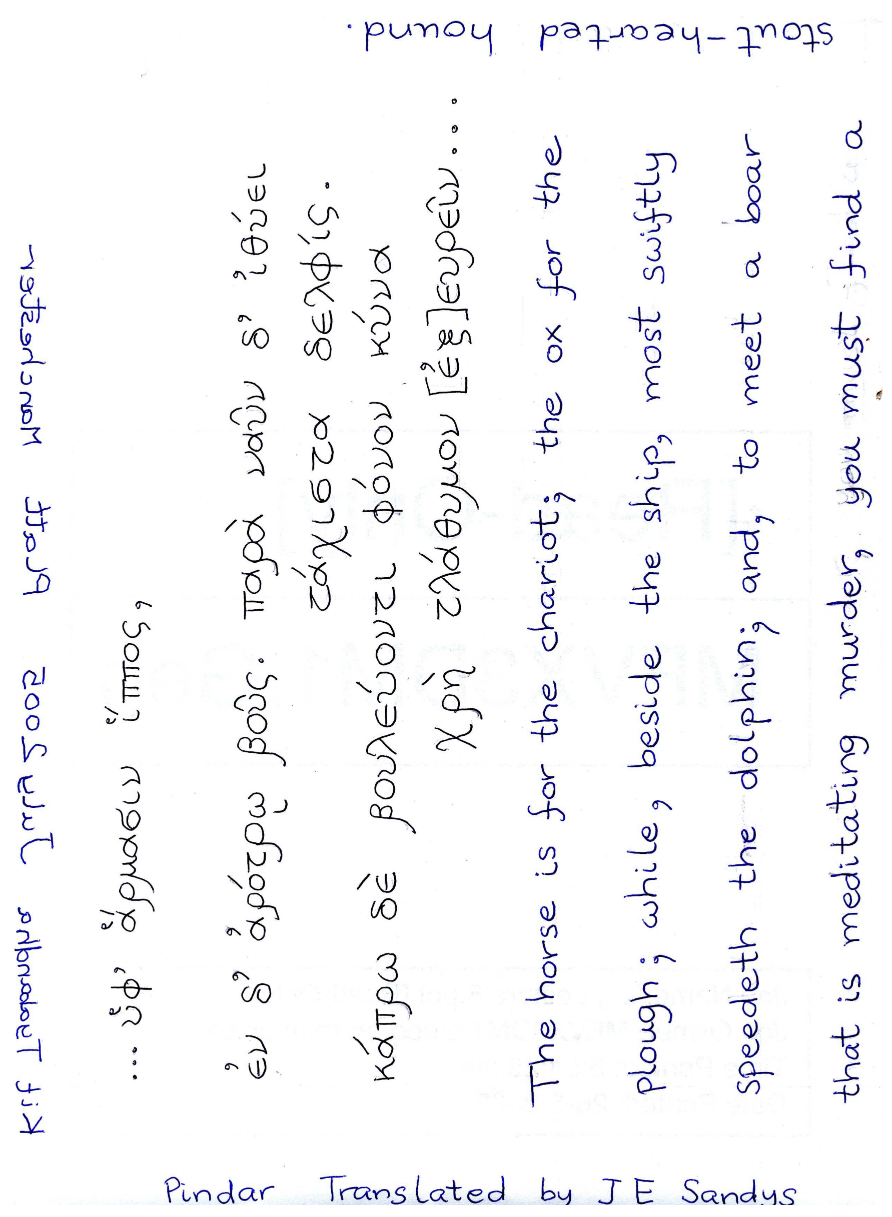


Kit Tyabandha July 2007

Ἀρχὰ μεγάλας ἀρετᾶς, ὤνασσα
Ἀλάθεια, μὴ πταίσῃς ἐμὰν
σύνθεσιν τραχεῖ ποτὶ ψεύδει...

Queen of Truth, who art the beginning
of great virtue, keep my good-faith
from stumbling against rough falsehood.

Pindar Translated by J E Sandys



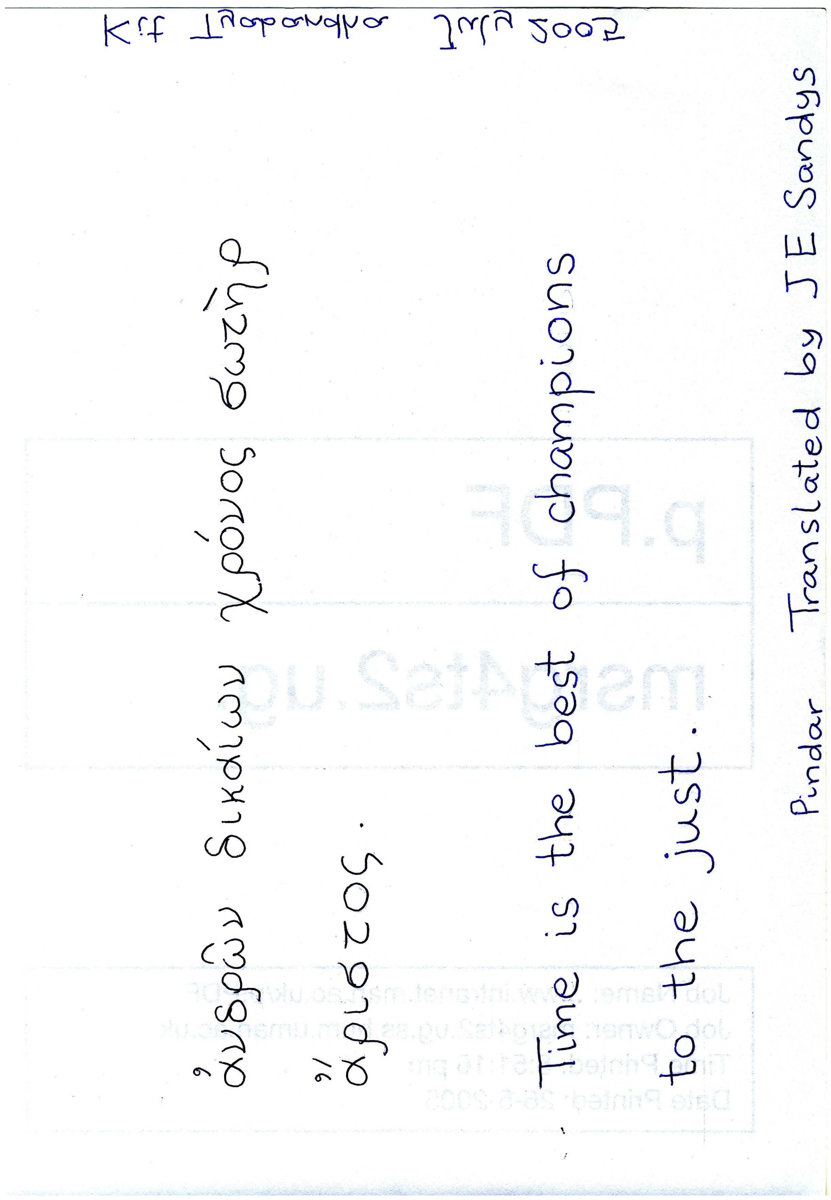
Kit Tyabandha July 2007 Draft Manchester

… ὑφ᾽ ἅρμασιν ἵππος,

ἐν δ᾽ ἀρότρῳ βοῦς. παρὰ ναῦν δ᾽ ἰθύει
τάχιστα δελφίς.
κάπρῳ δὲ βουλεύοντι φόνον κύνα
χρὴ τλάθυμον [ἐξ]ευρεῖν…

The horse is for the chariot; the ox for the plough; while, beside the ship, most swiftly speedeth the dolphin; and, to meet a boar that is meditating murder, you must find a stout-hearted hound.

Pindar Translated by J E Sandys

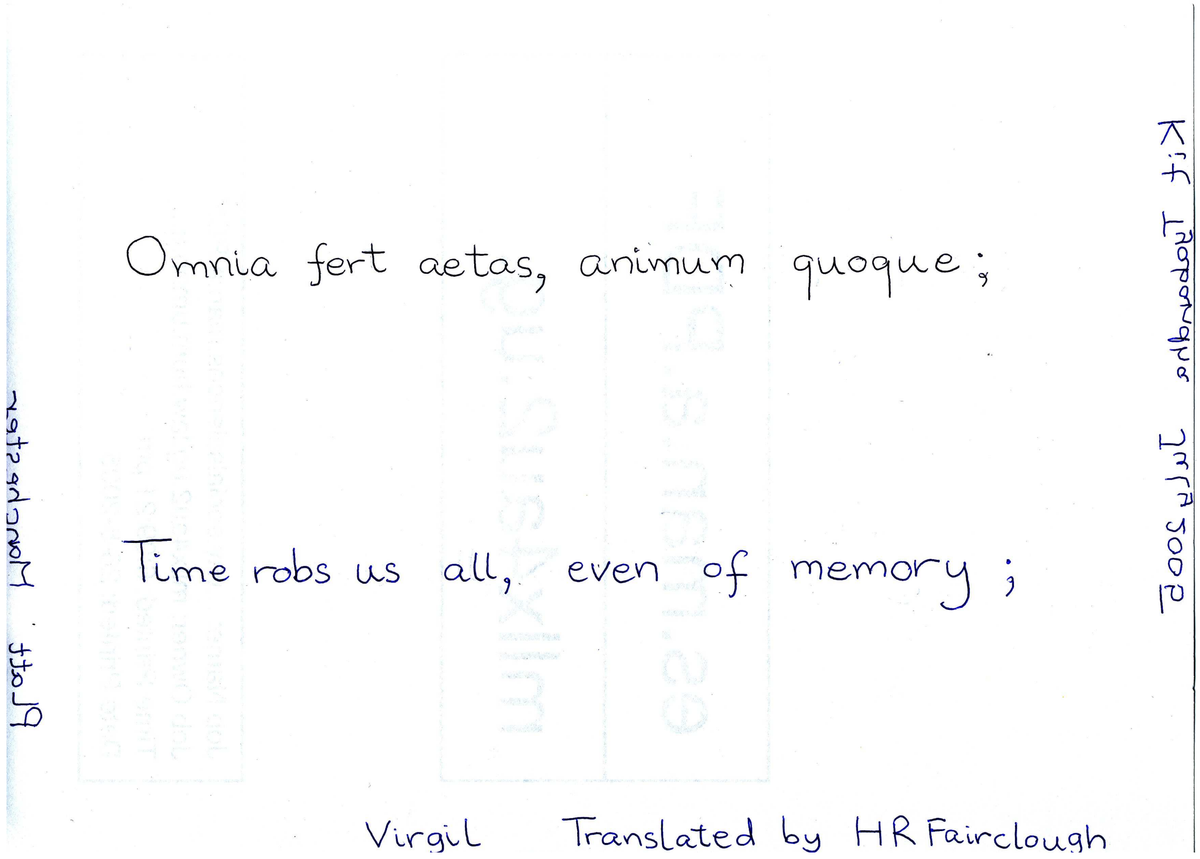


Kit Tyabandha July 2007

ἀνδρῶν δικαίων χρόνος σωτὴρ
ἄριστος.

Time is the best of champions
to the just.

Pindar Translated by J E Sandys



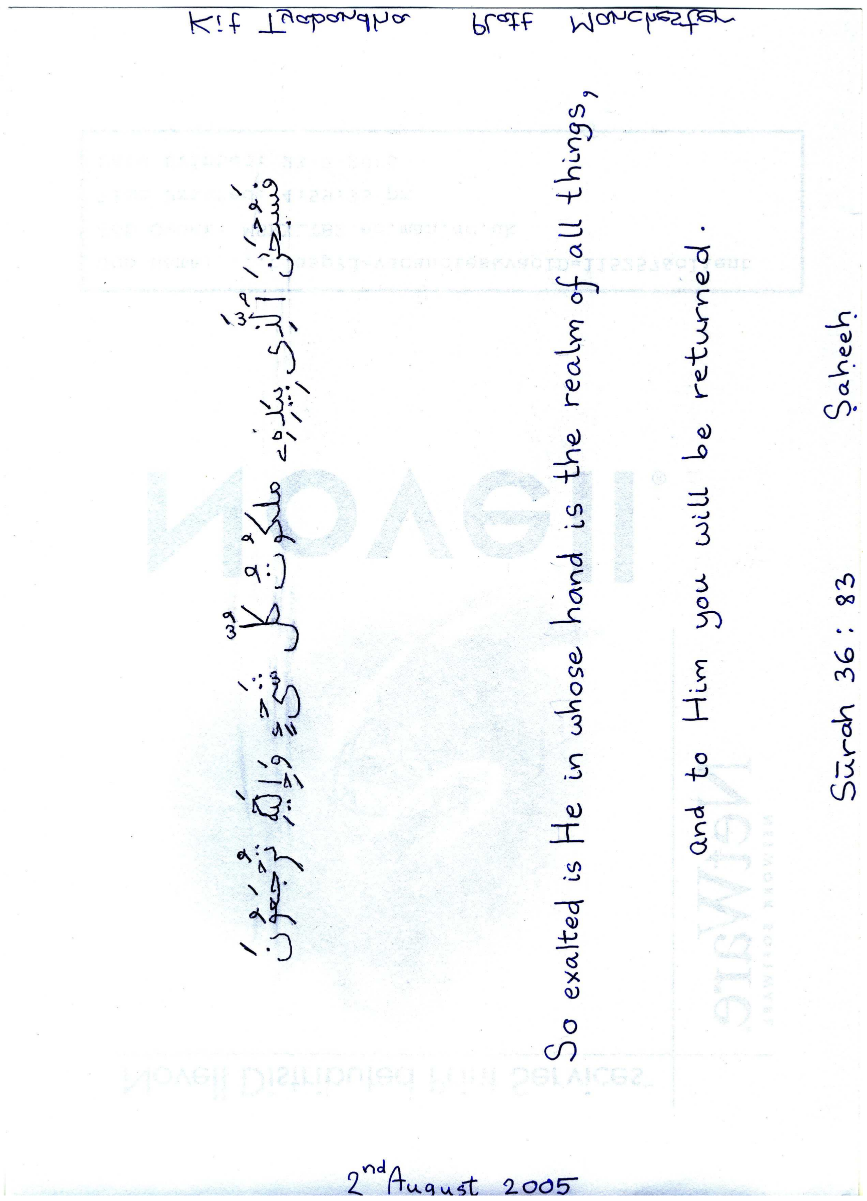
Kit Tapodhan Jul 2002

Omnia fert aetas, animum quoque;

Time robs us all, even of memory;

Virgil Translated by HR Fairclough

Platt Manchester



Kit Tyabandha Draft Manchester

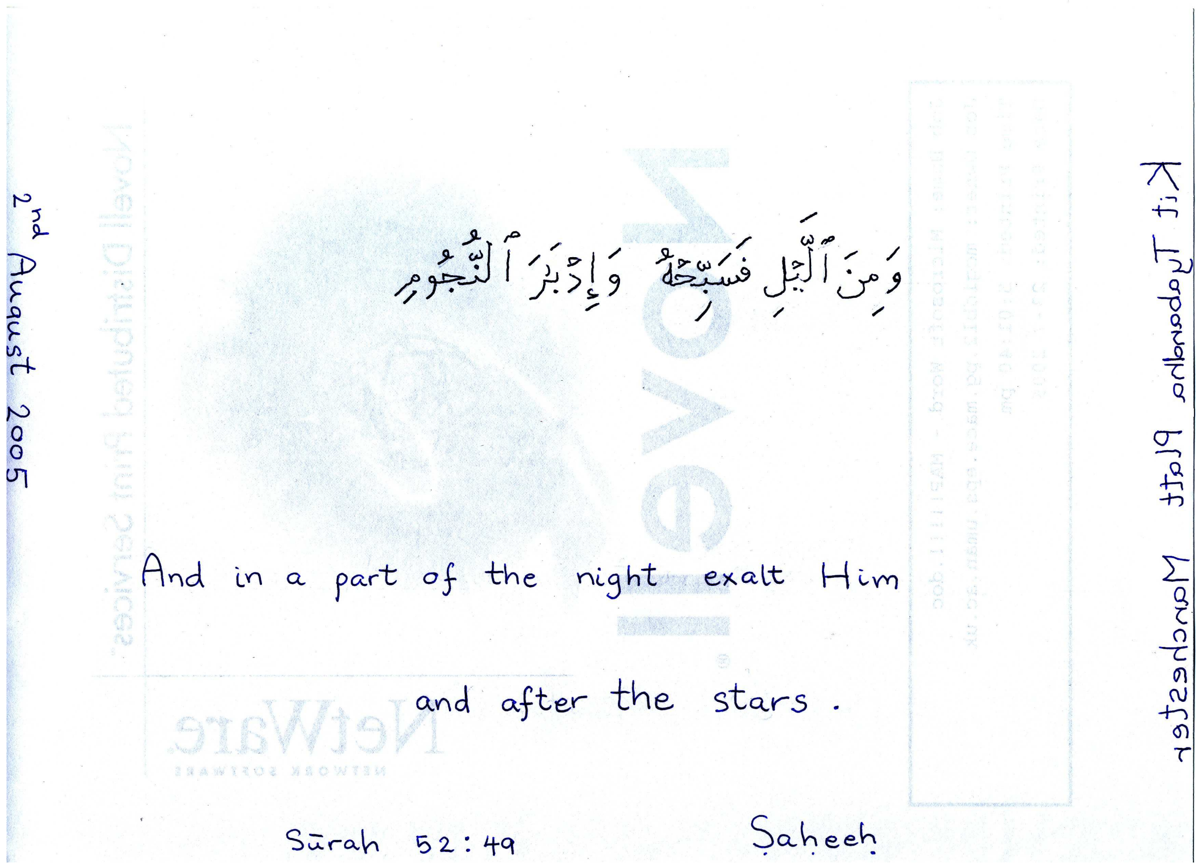
فَسُبْحَانَ الَّذِي بِيَدِهِ مَلَكُوتُ كُلِّ شَيْءٍ وَإِلَيْهِ تُرْجَعُونَ

So exalted is He in whose hand is the realm of all things,

and to Him you will be returned.

Sūrah 36 : 83 Saheeh

2nd August 2005



Kif Tyabandha Platt Manchester

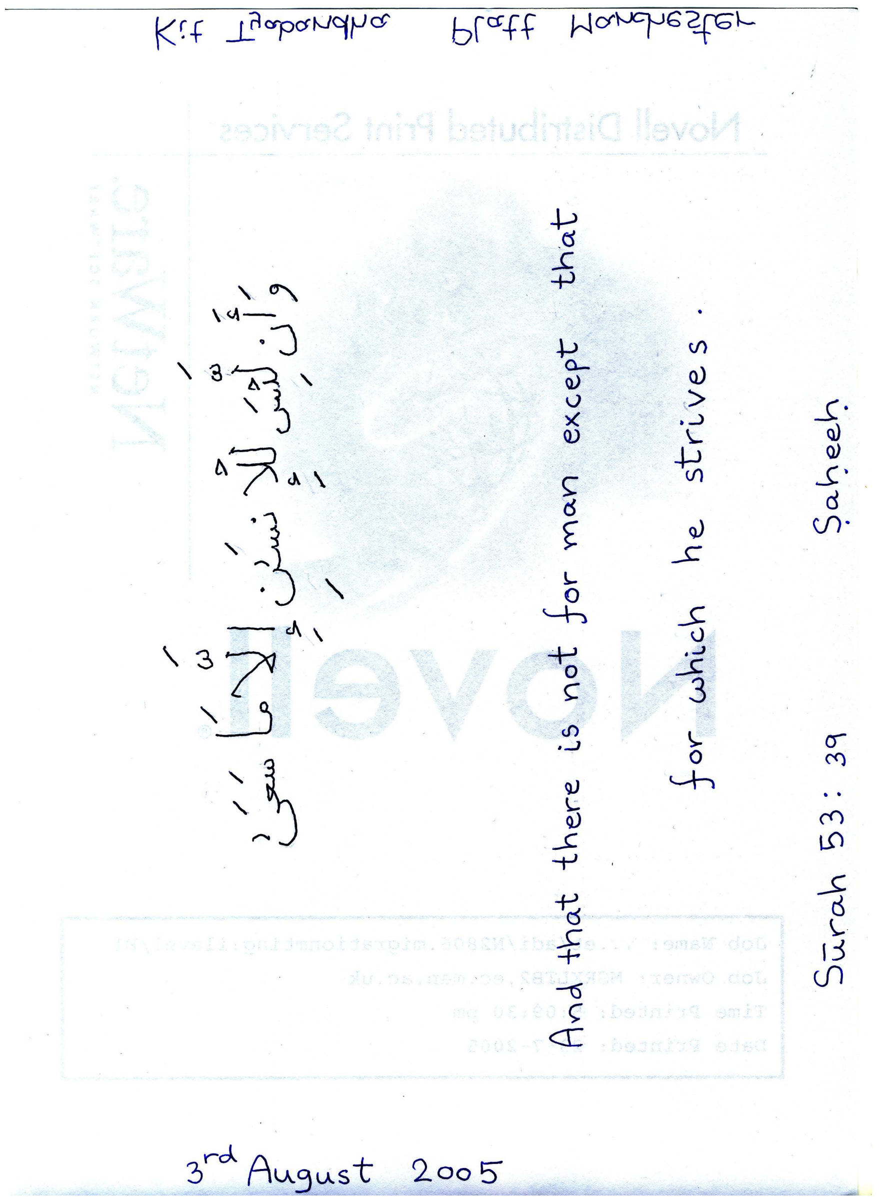
وَمِنَ ٱلَّيۡلِ فَسَبِّحۡهُ وَإِدۡبَٰرَ ٱلنُّجُومِ

And in a part of the night exalt Him

and after the stars.

Sūrah 52:49 Ṣaḥeeḥ

2nd August 2005



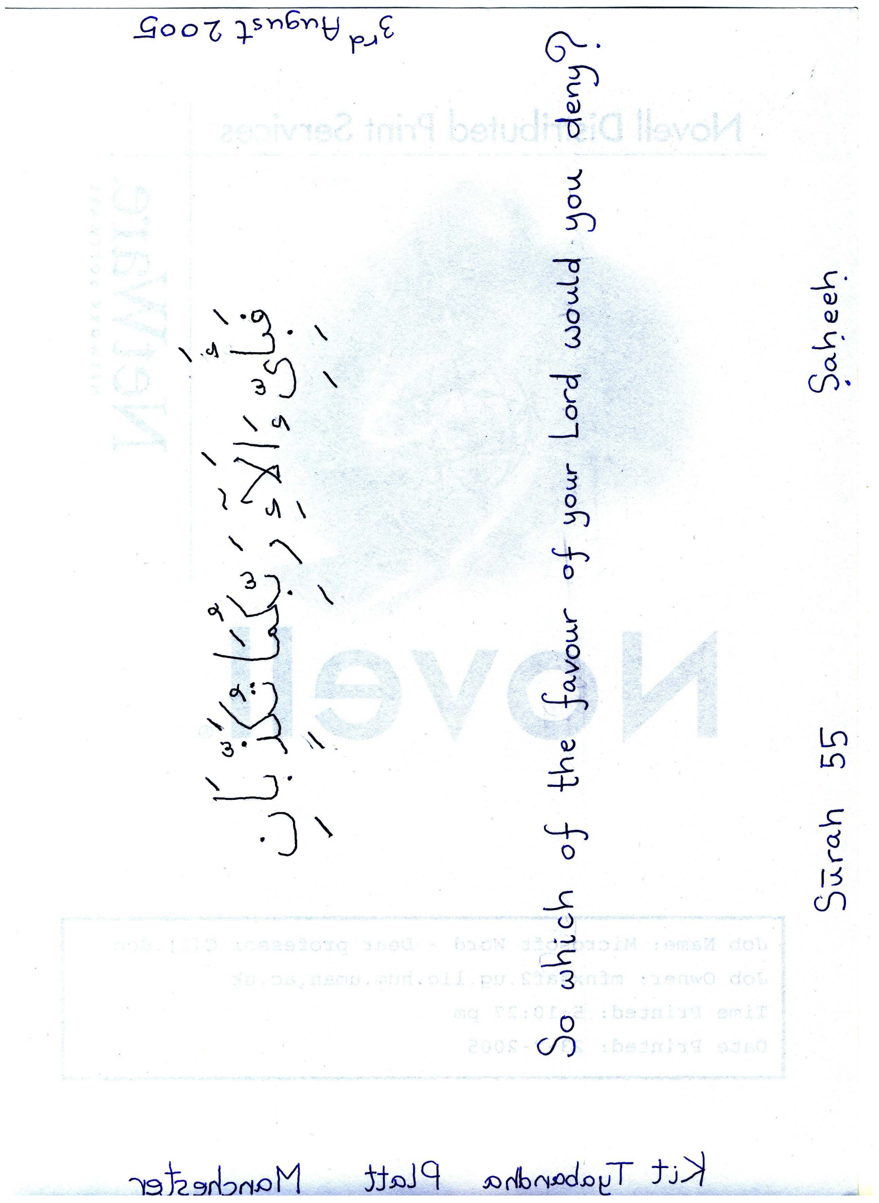
Kit Tapandhua Draft Manchester

وَأَن لَّيْسَ لِلْإِنسَٰنِ إِلَّا مَا سَعَىٰ

And that there is not for man except that for which he strives.

Sūrah 53 : 39 Ṣaḥeeḥ

3rd August 2005



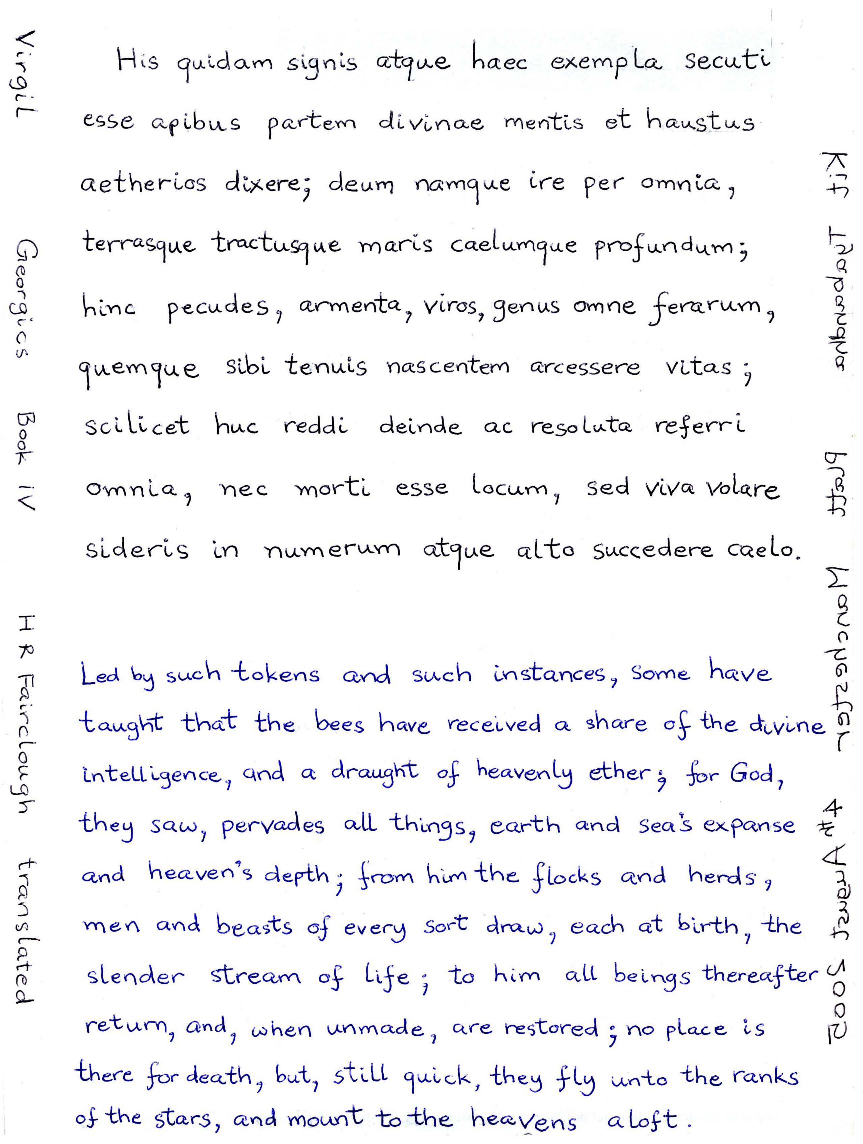
3rd August 2005

فَبِأَيِّ آلَاءِ رَبِّكُمَا تُكَذِّبَانِ

So which of the favour of your Lord would you deny?

Sūrah 55

Saḥeeḥ



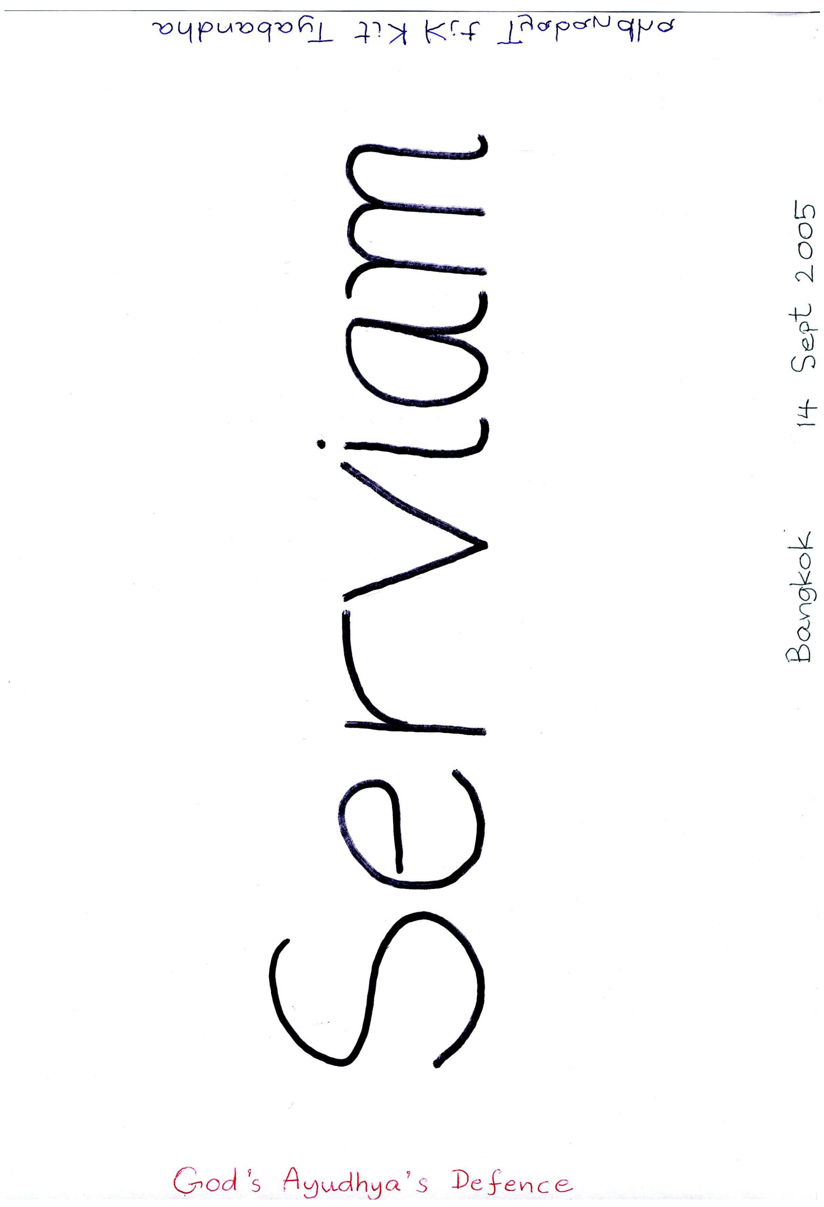
Virgil Georgics Book IV

His quidam signis atque haec exempla secuti
esse apibus partem divinae mentis et haustus
aetherios dixere; deum namque ire per omnia,
terrasque tractusque maris caelumque profundum;
hinc pecudes, armenta, viros, genus omne ferarum,
quemque sibi tenuis nascentem arcessere vitas;
scilicet huc reddi deinde ac resoluta referri
omnia, nec morti esse locum, sed viva volare
sideris in numerum atque alto succedere caelo.

H R Fairclough translated

Led by such tokens and such instances, some have taught that the bees have received a share of the divine intelligence, and a draught of heavenly ether; for God, they saw, pervades all things, earth and sea's expanse and heaven's depth; from him the flocks and herds, men and beasts of every sort draw, each at birth, the slender stream of life; to him all beings thereafter return, and, when unmade, are restored; no place is there for death, but, still quick, they fly unto the ranks of the stars, and mount to the heavens aloft.

Kit Tyabandha draft Manchester 4 August 2005

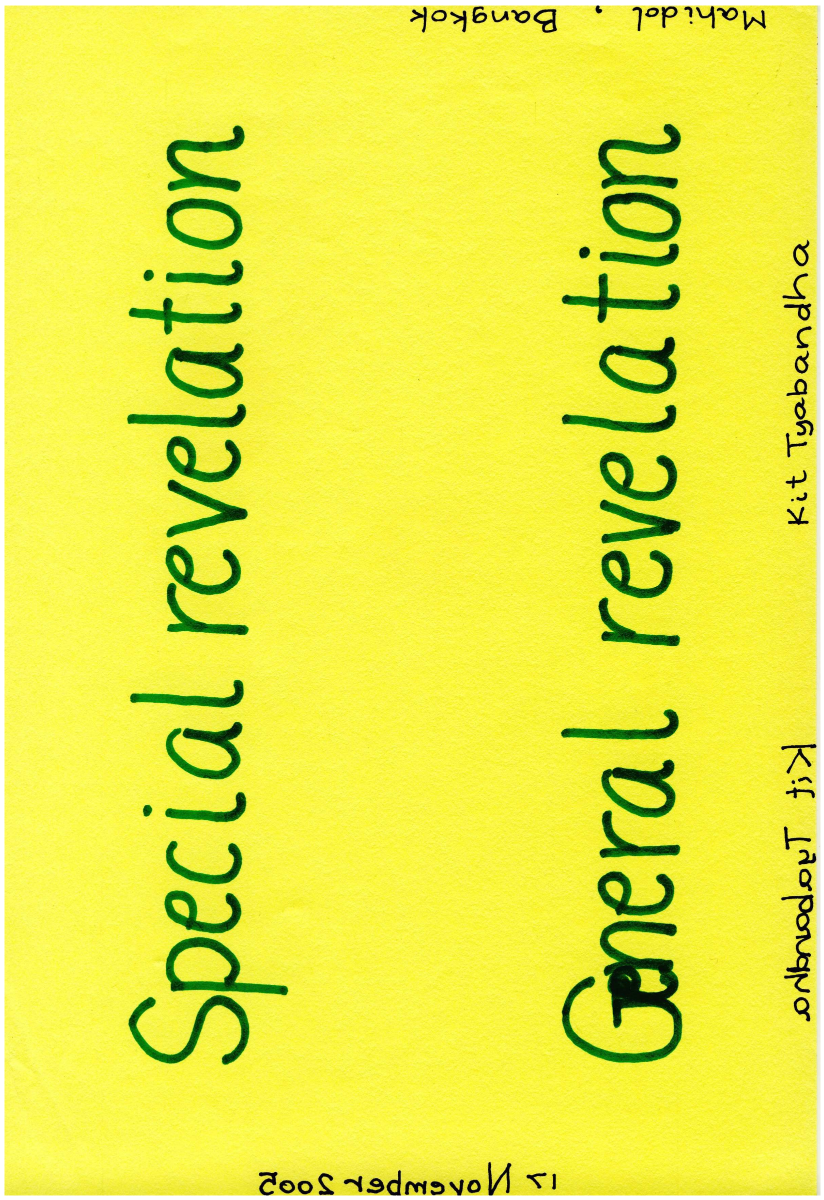


Kit Tyabandha

# Serviam

Bangkok 14 Sept 2005

God's Ayudhya's Defence



Special revelation

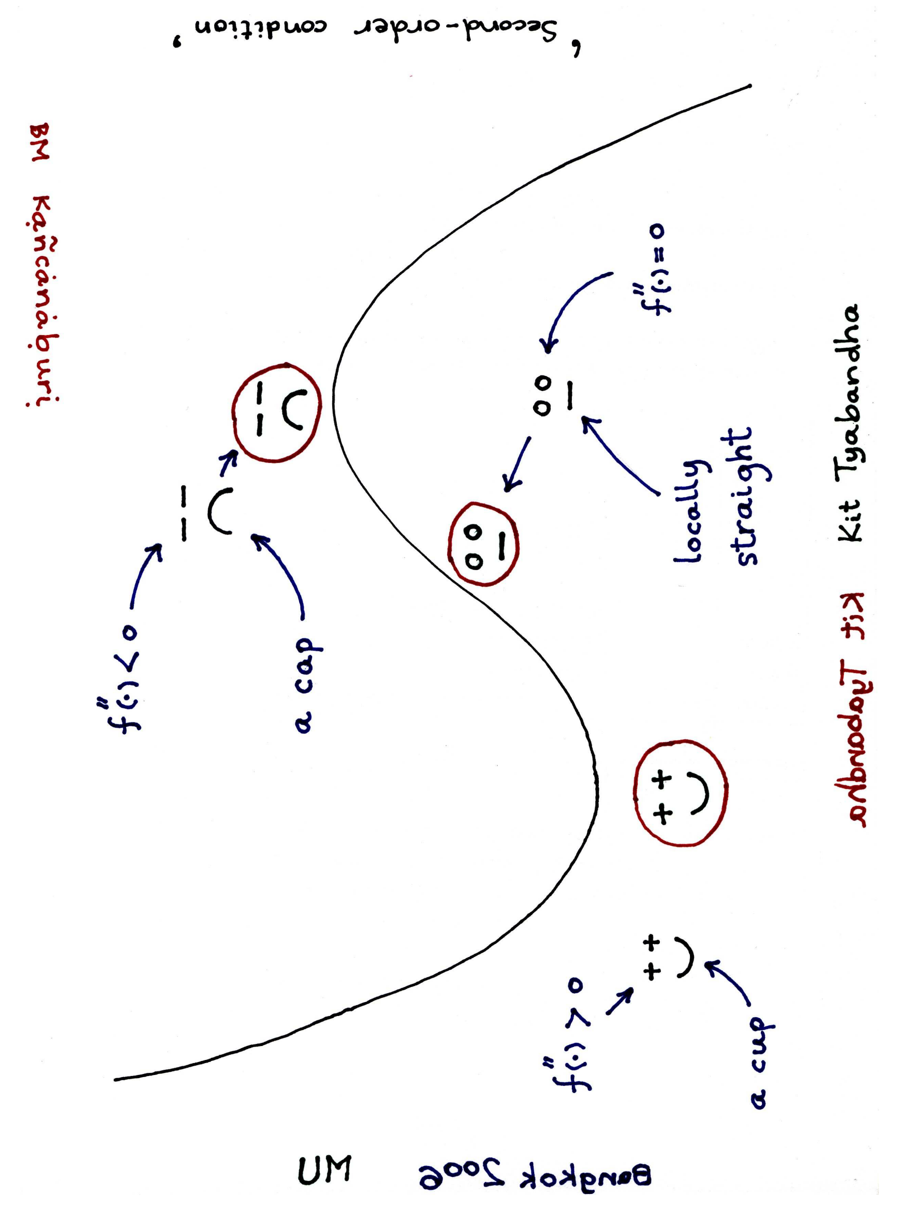
General revelation

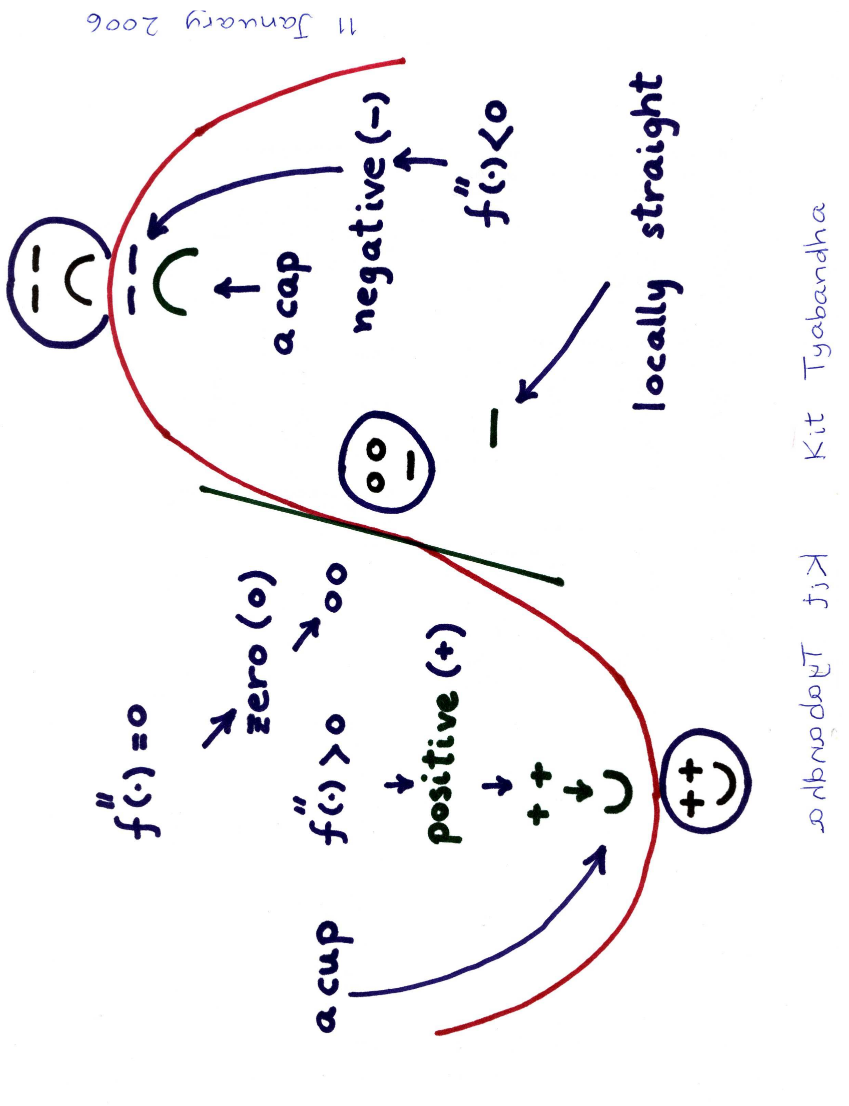
Kit Tyabandha

Mahidol, Bangkok

17 November 2005

# Sabbath


'Second-order condition'
BM Kañcanaburi
$f''(\cdot) < 0$
a cap
$f''(\cdot) = 0$
locally straight
Kit Tyabandha
Kit Tyabandha
$f''(\cdot) > 0$
a cup
Bangkok 2006
MU
UM


11 January 2006
a cap
negative (−)
$f''(\cdot)<0$
locally straight
$f''(\cdot)=0$
zero (0)
$f''(\cdot)>0$
positive (+)
a cup
Kit Tyabandha

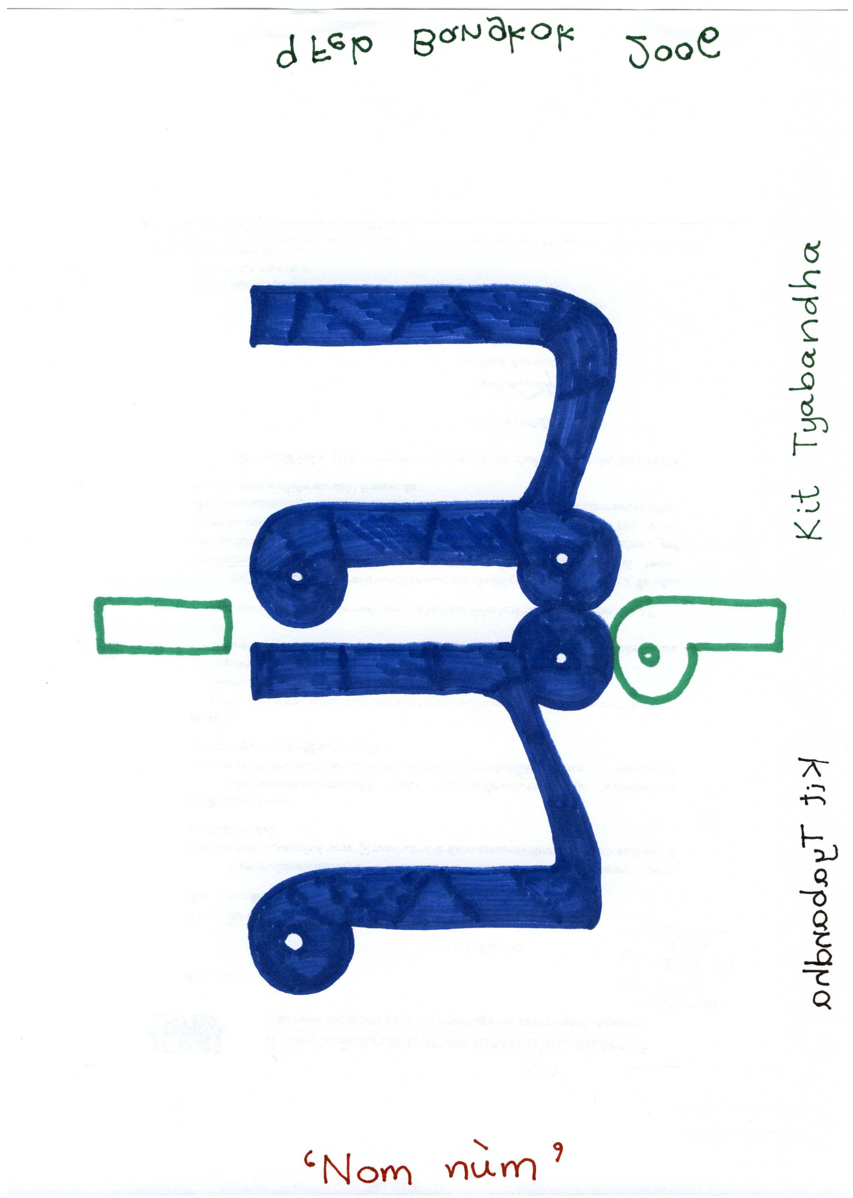
12 January 2006

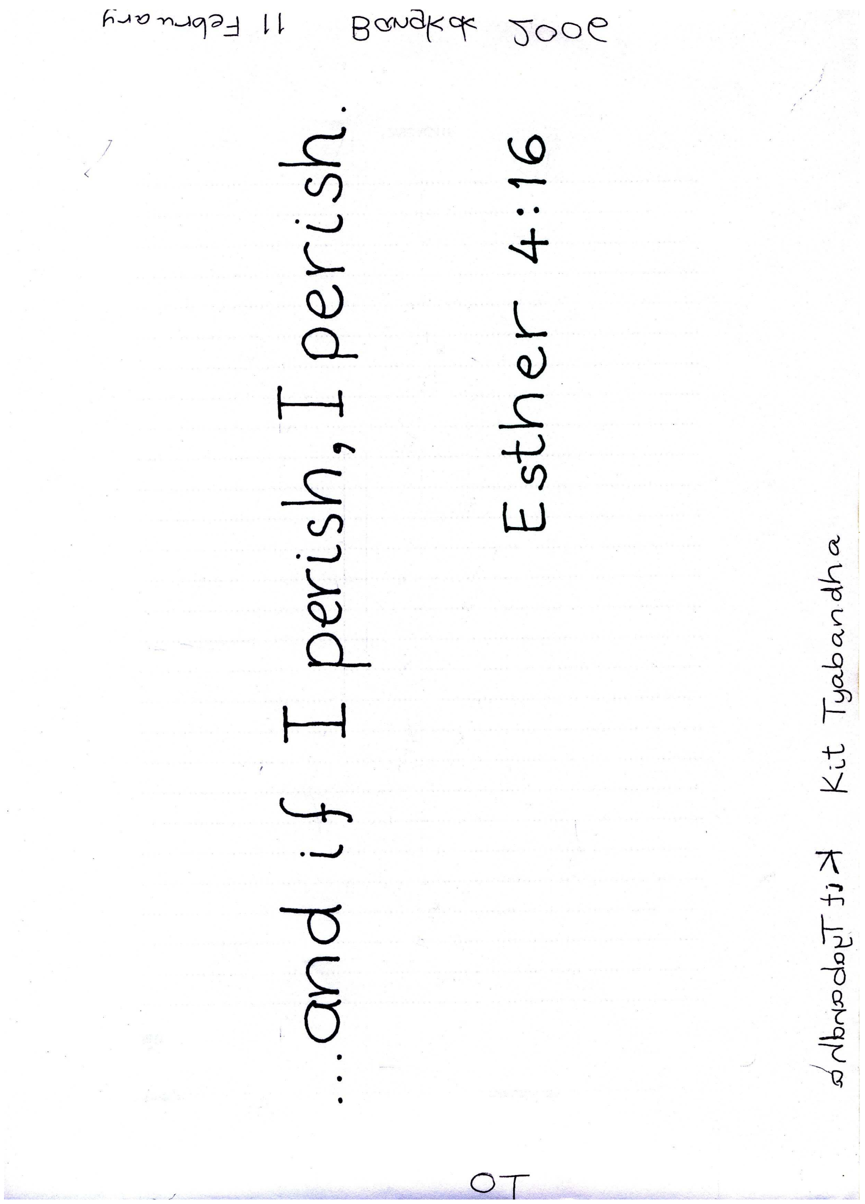
目が閉して、胸の中
悲しつづけるなければ

'Experience'

Bangkok 2006

Kit Tyabandha Kit Tyabandha


Kit Tyabandha
Kit Tyabandha
'Nom nùm'



2007 Bangkok 11 February

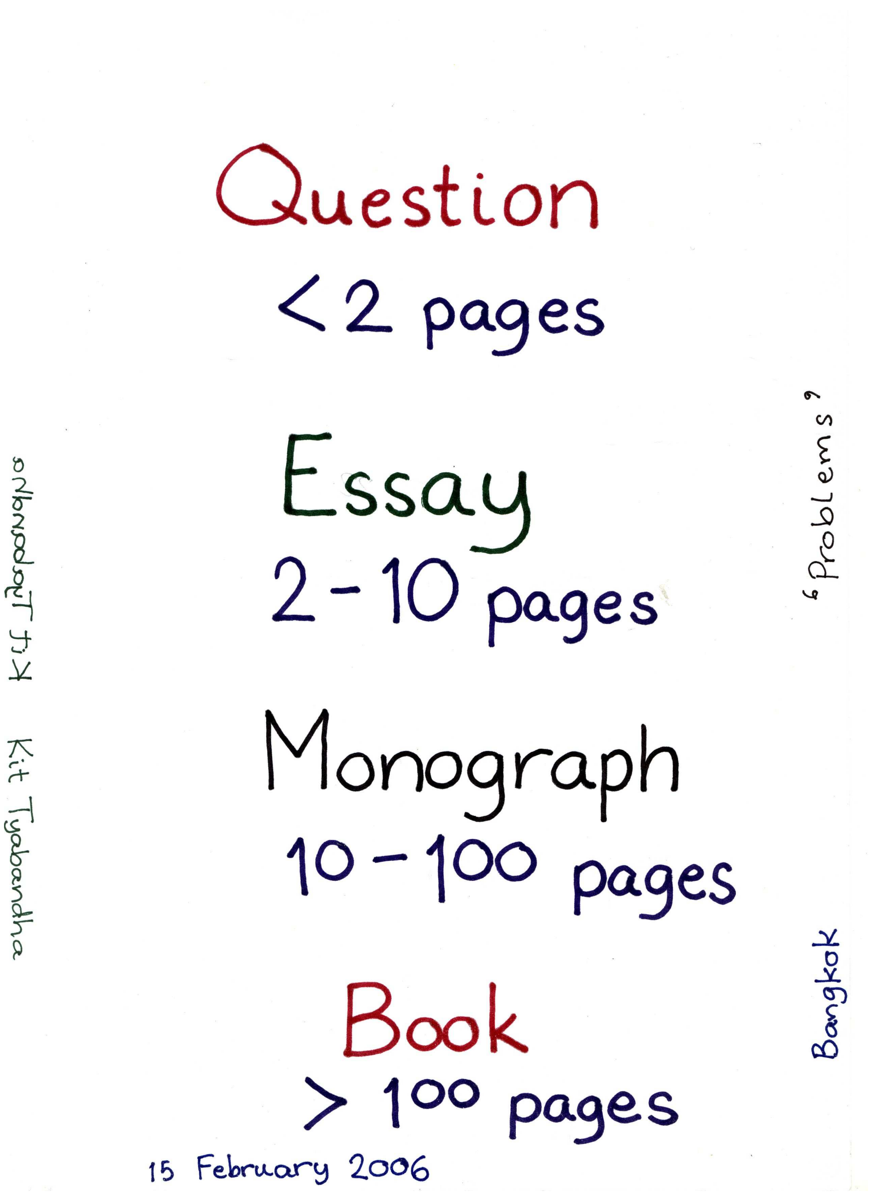
...and if I perish, I perish.

Esther 4:16

Kit Tyabandha

Kit Tyabandha

OT



Question

< 2 pages

Essay

2 - 10 pages

Monograph

10 - 100 pages

Book

> 100 pages

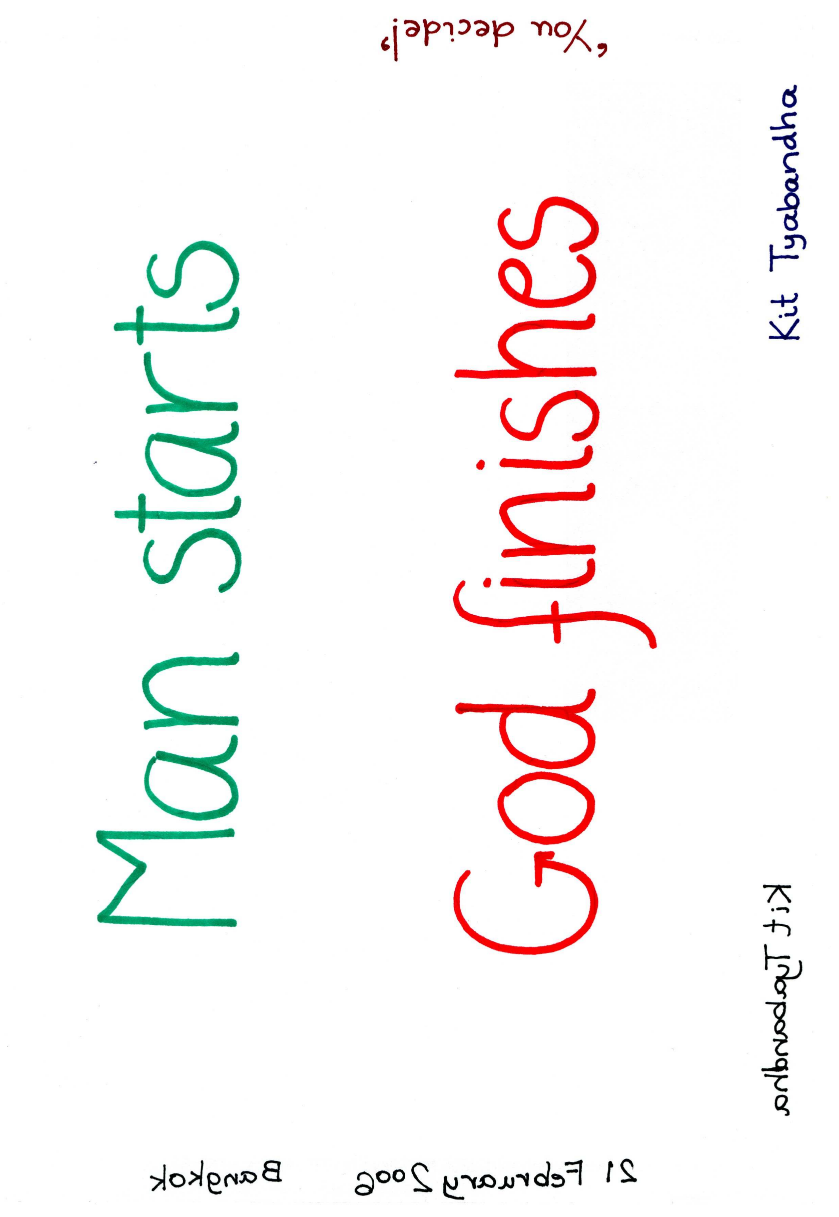
15 February 2006

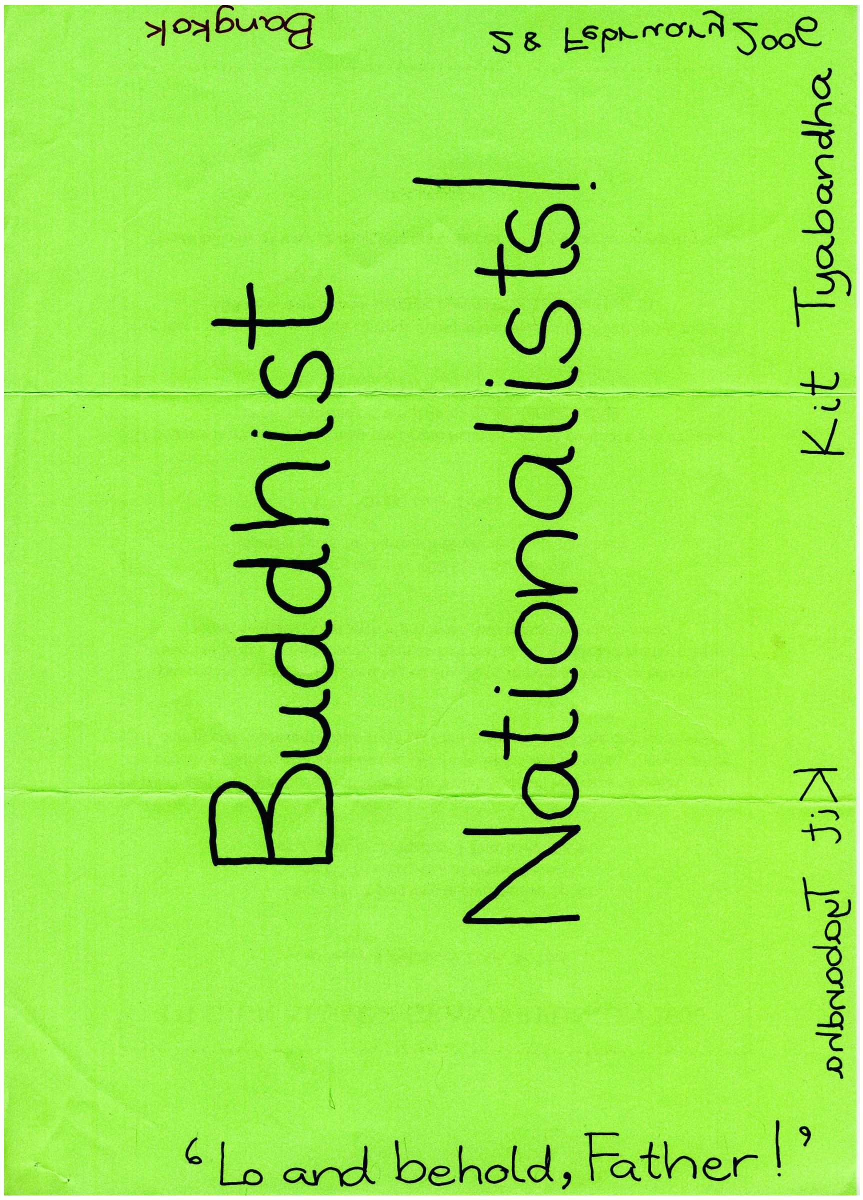
Kit Tyabandha

Kit Tyabandha

'Problems'

Bangkok


'You decide!'
Man starts
God finishes
Kit Tyabandha
Kit Tyabandha
21 February 2006 Bangkok

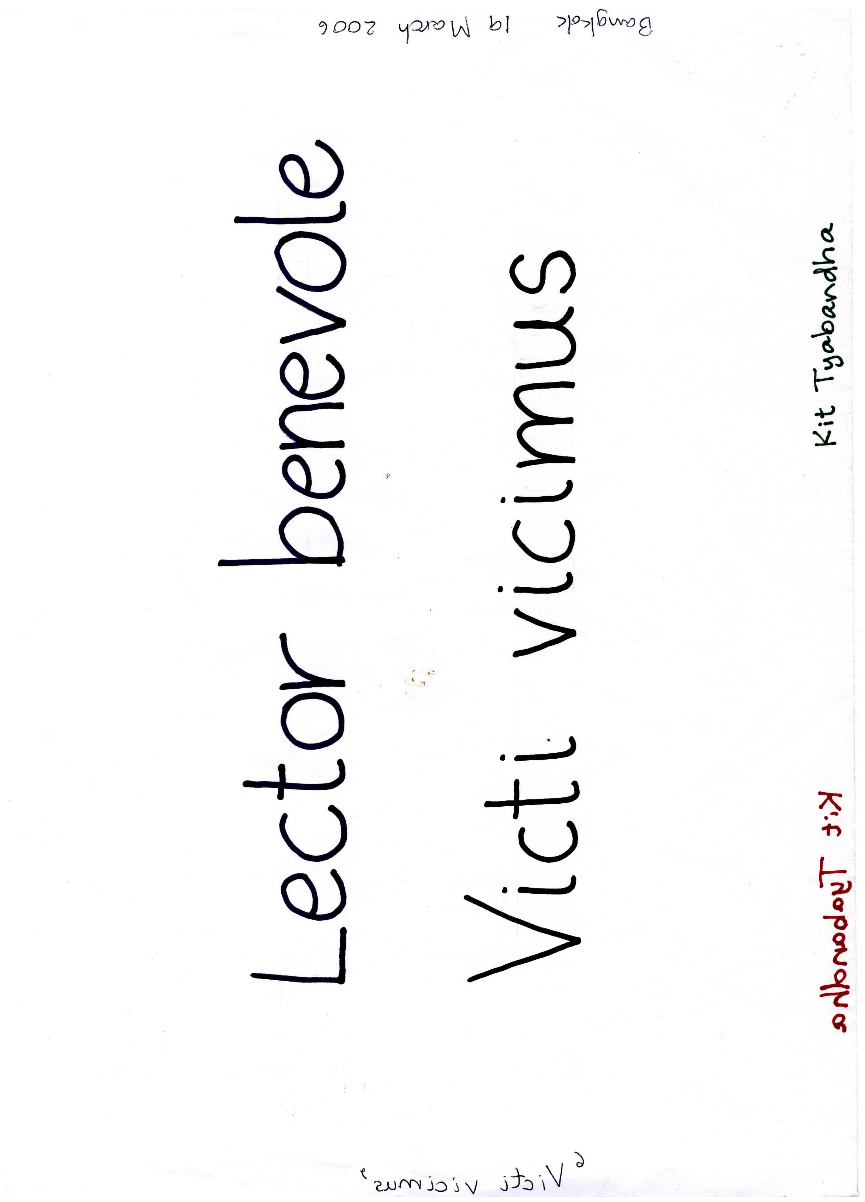


Bangkok

Buddhist Nationalist!

Kit Tyabandha

'Lo and behold, Father!'



Lector benevole

Victi vicimus

Kit Tyabandha

6 Jan 2007

No fors though I spille!

Kit Tyabandha

Kit Tyabandha

' To perish '